VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Platforma pro distribuci iOS aplikací
Platform for Distribution of iOS Applications

2017 Libor Polehňa

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Libor Polehňa**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Platforma pro distribuci iOS aplikací
Platform for Distribution of iOS Applications

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je vytvoření systému pro usnadnění instalací a sdílení aplikací na platformě iOS mimo oficiální obchod App Store. Standardní App Store nenabízí snadnou distribuci ještě nevydaných aplikací. Aplikace musí projít schvalovacím procesem, který má svá pravidla a limity. Je potřeba vyvinout platformu, která odstraní a zrychlí sdílení nevydaných aplikací mimo App Store.

1. Analýza stávajících řešení, identifikace klíčových nedostatků a návrh nových funkcí, které tyto nedostatky řeší.
2. Analýza navrhovaného řešení, sběr požadavků, nalezení klíčových případů užití.
3. Návrh architektury řešení. A to s důrazem na flexibilitu řešení ve vztahu k přidání nových funkcí a změnám použitých technologií.
4. Implementace a nasazení řešení. Implementace je klíčovou částí práce, a proto bude kladen důraz na funkčnost řešení a dodržení dobrých praktik programování. Výsledné řešení by mělo být nasazeno na server.
5. Vypracování dokumentace řešení, která bude obsahovat dokumentaci kódu, návod pro nasazení a údržbu systému.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Vladislav Skoumal**

Konzultant bakalářské práce: Ing. Jan Plucar

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky 1D16 OSTRAVA

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

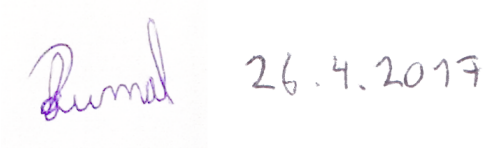
Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

26. 4. 2017 Poletín

Prohlášení zástupce

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.



Abstrakt

Cílem bakalářské práce je vytvořit systém pro distribuci iOS aplikací mimo oficiální obchod App Store. Systém se skládá ze tří částí serverový backend, iOS aplikace a macOS aplikace. Cíl systému je zjednodušit a vylepšit existující řešení na trhu. Hlavní výhody oproti jiným řešením jsou jednoduché přihlášení, jednoduchý design, historie verzí a možnost brandování aplikací. Práce obsahuje analýzu trhu, sběr požadavků, návrh řešení, implementaci systému a nasazení do provozu.

Klíčová slova

iOS, runtime, distribuce, macOS, Apple, vývoj software

Abstract

The objective of the bachelor thesis is to create system for distribution iOS applications outside of official store App Store. System is consists of three parts server backend, iOS application and macOS application. The system aims to simplify and improve existing solutions in the market. Main advantages compared to others systems are simple login, simple design, branding, version history and possibility of branding applications. Thesis includes market analysis, requirements gathering, solution design, implementation and deployment of the system.

Keywords

iOS, runtime, distribution, macOS, Apple, software development

Obsah

0.1 Seznam použitých symbolů a zkratek

REST Representational State Transfer
API Application Programming Interface
UUID Universally Unique Identifier
HTTP Hypertext Transfer Protocol
HTTPS Hypertext Transfer Protocol Secure
CI Continous Integration
SDK Software Development Kit
UML Unified Modeling Language
ID Identifier
MVT Model - View - Template
XML eXtensible Markup Language
JSON JavaScript Object Notation
MVC Model - View - Controller
MVVM Model - View - ViewModel
ORM Object-Relational Mapping
WWW World Wide Web

0.2 Seznam ilustrací a seznam tabulek

Seznam obrázků

Seznam tabulek

1 Úvod

Sdílení aplikací na platformě iOS mimo App Store je složitá problematika a existující služby na trhu nenabízejí jednoduché řešení. Ještě než se aplikace dostane do App Store je potřeba ji testovat v určitém okruhu lidí. Každá aplikace prochází při vývoji určitými fázemi vývoje označované jako Alpha, Beta a Release. Je potřeba přijít s jednoduchým řešením, jak zákazníkům a testerům sdílet aplikaci.

Již existující řešení na trhu jsou složité, či nekomfortní, a tak nás to dovedlo si vymyslet a zrealizovat vlastní řešení, které bude pohodlné pro nás a hlavně jednoduché ze strany zákazníka. Naše řešení se skládá ze tři hlavních částí: Mobilní aplikace, macOS správce systému a serverový backend. Mobilní aplikace slouží k instalaci, mazání a updatování aplikací. macOS správce slouží k nahrávání aplikací do systému. Poslední část systému je serverový backend, který zajišťuje přístup k datům.

Práce je rozdělena do třech hlavních částí. První část se zabývá analýzou současného trhu, porovnání existujících řešení a nalezení jejich nedostatků. Druhá část obsahuje návrh nového řešení, sběr požadavků, nalezení případů užití a návrh uživatelského rozhraní. Poslední a zároveň hlavní část se zabývá samotnou technickou implementací řešení. Zde je kladen důraz na architekturu, funkčnost a samotné nasazení systému do praxe. [1, 2]

2 Cíl práce

Cílem práce je analýza existujících řešení na trhu a nalezení jejich nedostatků a následné odstranění, zjednodušení a vylepšení existující funkčnosti.

Řešení obsahuje mobilní aplikaci, která se doručí zakázníkům a testerům, ze které budou instalovat, mazat a updatovat ostatní aplikace. Mobilní aplikace bude podobná obchodu App Store.

Dále je zde desktopový klient pro macOS systém, který slouží pro správu aplikací a projektů v systému.

Další část je serverový backend, který zastřešuje přístup k datům a komunikaci s desktopovou a mobilní aplikaci přes REST API. Obsahuje taky jednoduché administrativní rozhraní na webu pro správu uživatelů v systému.

Nakonec bude rozebráno nasazení systému na server, jeho použití v praxi, údržbu a další vývoj.

3 Analýza trhu

3.1 Sdílecí metody

Instalace aplikací při vývoji mimo simulátor vyžaduje určitou konfiguraci. Konfiguraci můžeme rozdělit na tři části: Certifikáty, registrace identifikátorů a provozní profily.

Každá firma, nebo samotný vývojář musí mít vlastní certifikát, kterým se podepisuje aplikace při exportování výsledného IPA souboru.

Každá aplikace musí mít zaregistrovaný svůj identifikátor. Identifikátor je jedinečný v rámci všech existujících aplikací. Takový identifikátor se skládá vetšinou z reverzní domény. Příklad: com.example.myapp

Provozní profil je soubor, který obsahuje hlavně identifikátor existujícího certifikátu, identifikátor aplikace a povolený způsob distribuce. Provozní profil je vložen do aplikace při kompilaci.

3.1.1 Distribuce

Na platformě iOS Apple poskytuje dva technické způsoby jak sdílet aplikaci mimo App Store. Způsoby se liší typem, jak jde aplikaci sdílet a taky hlavně cenou.

První variantou a zároveň nejlevnější je metoda nazývána Ad Hoc. Metoda Ad Hoc vyžaduje registraci zařízení podle UUID. Registrace definuje povolené zařízení na kterých půjde aplikaci spustit. Po zaregistrování všech zařízení se do aplikace vloží provozní profil, který obsahuje UUID všech zařízení, které chceme podporovat. Následně se aplikace zkompiluje a vytvoří se IPA soubor, který je odeslán uživatelům. Ad Hoc tedy vyžaduje spolupráci testerů či zákazníka a vývojáře. Registrace nových zařízení je zdlouhavé a nekomfortní z pohledu zákazníka. Ad Hoc metoda je také omezena na maximálně 100 zařízení. Cena za tuhle službu se pohybuje kolem 100 dolarů za rok.

Druhou variantou je metoda nazývána In-House. Metoda In-House na rozdíl od Ad Hoc nevyžaduje registraci konkretních zařízení. In-House funguje na principu podepsání aplikace distribučním certifikátem. Výsledný IPA soubor lze sdílet na jakékoliv zařízení a není omezený počtem. Jedinou nutnou podmínkou, kterou zákazník, či tester musí udělat je ta, že při první instalaci aplikace potvrdí, že chce důvěřovat firmě, která podepsala IPA soubor svým distribučním certifikátem. Tahle metoda je náročná ze strany vývojáře, protože vyžaduje HTTPS server, ze kterého se bude aplikace stahovat resp. instalovat do zařízení. Nešifrovaný HTTP server není podporován a systém odmítne instalovat aplikaci z takového zdroje. Cena téhle služby se pohybuje kolem 300 dolarů za rok.

3.2 Existující služby

Existujících řešení na trhu je spoustu. Většina z nich nabízí stejnou funkcionalitu. Některé služby to kombinují s crash reporting službou a nebo taky s CI - continous integration. Při analýze současných řešení se primárně zaměříme pouze na sdílení aplikací.

3.2.1 Apple TestFlight

První věc, na kterou narazíme na platformě iOS je TestFlight [3]. Je to služba přimo od Apple. TestFlight poskytuje pouze sdílení aplikací. Sdílení aplikací může být přes dva způsoby. První způsob je Internal Testing. Interního testování se může zúčastnit pouze 25 uživatelů a každý uživatel může mít maximálně 10 zařízení. Druhý způsob je External Testing. Externího testování se může zúčastnit už 2 000 lidí. U Externího testování musí aplikaci projít schválením ze strany Applu. Obě dvě metody jsou omezeny na maximálně 100 aplikací. Pro přidání uživatele k testování je potřeba ho pozvat přes email. Další nevýhodou je to, že není možné sdílet verze Alpha, Beta a Release jedné aplikace. Vždy se jedná o release verzi. Jestli chceme distribuovat více verzí, tak pro každou verzi musíme vytvořit novou aplikaci v TestFlight systému. Tento způsob přináší další komplikace při distribuci aplikací. Výhodou TestFlight je, že má vlastní iOS aplikaci a uživatel si tak může na jednom místě spravovat všechny aplikace, které má nasdílené. TestFlight jako služba je zdarma, ale vyžaduje učet na iTunes Connect, který je placený a přístup do TestFlight služby probíhá přes něj.

3.2.2 HockeyApp

HockeyApp je služba od Microsoftu [4]. Kromě samotného sdílení aplikací nabízí další služby navíc, jako crash reporting a analytics. HockeyApp podporuje obě možnosti distribuce (Ad Hoc a In-House). Nevýhodou je potřeba naimportovat HockeyApp SDK do projektu při vývoji. Důvod, proč je to bráno jako nevýhoda, je ten, že se jedná pouze o službu, která distribuuje již hotové aplikace a není zde potřeba zahrnovat další knihovnu od třetí strany. Jelikož existující řešení na trhu nabízejí k samotné distribuci i více služeb, tak nutí vývojáře zahrnout jejich SDK do projektu, i bez využití jiných služeb. Další nevýhodou je pricing model, který začíná zdarma s omezením na pouze dvě aplikace. Za limit 15 aplikací zaplatíte 30 dolarů za měsíc. Velká výhoda je že služba nabízí vlastní iOS aplikaci a taky webové rozhraní pro instalaci aplikací. Pro vývojáře mají vlastní macOS program pro sdílení aplikací. Jedná se o jednu z nejvíce používaných služeb.

3.2.3 Crashlytics Beta

Primární funkcí Crashlytics je crash reporting [5]. Vedle toho taky nabízejí sdílení aplikací. Podporují continous integration, Ad Hoc a In-House distribuci. Nevýhodou je potřeba naimportovat jejich vlastní SDK do projektu, důvod je stejný, který byl už zmíněn předtím. Uživatele je nutné pozvat do služby. Mají vlastní aplikaci pro macOS na správu aplikací a iOS aplikaci pro uživatele. Velkou výhodou je, že služba nemá žádné omezení na počet aplikací, uživatelů,

úložiště atp. Je zcela nelimitovaná a úplně zdarma. Služba nemá problém s distribucí více verzí aplikace (Alpha, Beta a Release). Jedná se o jednu z nejvíce používaných služeb.

3.2.4 Beta Family Supersend

Služba nabízí jednoduché sdílení aplikací. Vedle toho taky poskytuje komplexní testování aplikace skupinou uživatelů po celém světe. Funkce na sdílení aplikací je zcela zdarma a není potřeba ani registrace. Poskytuje pouze webové rozhraní, kde se vloží IPA soubor a následně se vygeneruje odkaz a QR kód. Informace jsou odeslány uživateli na email, který obsahuje odkaz na aplikaci a může si ji stáhnout. Služba jinak nenabízí nic jiného. Nemá vlastní aplikaci na iOS ani macOS. Její největší nevýhodou je, že nahraná aplikace je dostupná ke stažení pouze tři dny.

3.2.5 TestFairy

Služba mimo distribuce aplikací nabízí crash reporting, continous integration, logy a nahrávání obrazovky na zařízení. Distribuce je dostupna jak Ad Hoc, tak In-House. Pro uživatele poskytují webové rozhraní na instalaci aplikací a taky mají vlastní iOS aplikaci. Uživatel musí být ve službě registrován. Pouze pro sdílení aplikací není potřeba používat jejich SDK. Funkce na sdílení aplikací je zdarma.

3.2.6 Appaloosa

Jedna z mála aplikací, která se čistě zabývá distribucí aplikací [6]. Navíc mají pouze analytics. Podporují continous integration a spoustu dalších vývojářských nástrojů. Mají rozsáhlou správu uživatelů, kde jde nastavit skupiny, samotné uživatelé a k nim jaké aplikace budou dostupné. Nevýhodou zde je opět potřeba použít jejich vlastní SDK v projektu, důvod je stejný, který byl už zmíněn předtím. Mají vlastní aplikaci dostupnou pro iOS, kde uživatel může spravovat své aplikace. Jejich aplikace je hodně podobná App Store. Obsahuje hodnocení, screenshoty a popis jednotlivých aplikací. Nahrávání aplikací probíhá přes webové rozhraní nebo přes nějaký vývojářský nástroj, jako například Jenkins, Fastlane, nebo dokonce plugin do vývojového prostředí. Pro jednu aplikaci a 5 uživatelů je služba zdarma. Za více si připlatíte podle počtu uživatelů a aplikací.

3.2.7 AppBlade

Další aplikace na sdílení [7]. Mímo jiné taky obsahuje crash reporting a analytics. Podporuje Ad Hoc a In-House distribuci. Má možnost třídit uživatele do skupin. Pro správu aplikací mají webové rozhraní a pro uživatele vlastní iOS aplikaci. Je taky možnost nahrávat aplikace do systému pomoci REST API. Je potřeba použít jejich SDK v projektu. Podporují více verzí aplikací (Alpha, Beta a Release). Omezení mají na 25 zařízení a pří překročení limitu se musí za službu platit.

3.2.8 BirdFlight

Služba, která řeší hlavně distribuci aplikací [8]. Maji taky crash reporting, když využijete jejich SDK v projektu. Podporují Ad Hoc a In-House distribuci. Mají pouze webové rozhraní a to jak pro správu uživatelů, tak i pro instalaci aplikací a taky pro nahrávání nových verzí. Uživatele je nutné do služby pozvat. Služba je velmi jednoduchá. Všechny funkce jsou zdarma bez omezení.

3.2.9 Installr

Tahle služba se snaží udělat distribuci co nejsnadnější a nejrychlejší [9]. Podporují Ad Hoc a In-House distriuci. U Ad Hoc se snaží o co největší automatizaci při registrování UUID zařízení. Neposkytují vlastní iOS aplikaci, mají pouze webové rozhraní jak pro správu uživatelů, tak i pro instalaci aplikací. Pro nahrání nové verze do systémů poskytují command line API. Je to jedna z mála služeb, kde není potřeba registrace pro uživatel, kteří pouze testují aplikaci. Při použití služby zdarma jste omezeni na tři zařízení. Další zařízení se dokupují k jednotlivým aplikacím a jednorázově.

3.3 Shrnutí

Jak je vidět z analýzy existujících řešení, tak většina nabízí stejnou funkcionalitu a spíše přidávají funkcionalitu, která už nesouvisí s distribucí aplikací.

Všechny dostupné řešení, které mají vlastní iOS a nebo i macOS aplikaci, vyžadují registraci uživatele. Osobně tohle považuji za největší nedostatek, že se uživatel musí zabývat určitou službou a registraci na ni. Důvod proč je registrace brána jako nevýhoda je ten, že není ve většině případů nutné nutit zákazníka, nebo testera, aby se registroval do služby. Pro velkou skupinu uživatelů může být registrace, nebo pozvání jednotlivých uživatelů do služby časově náročný proces. Všechny služby, které jsou nabízeny zároveň s distribucí, jako například: odesílání crashů a identifikace jednotlivých uživatelů, lze obejít službami, které se pouze zabývají danou problematikou.

Další z nevýhod můžeme označit, u některých služeb, nutnost zahrnout do projektu jejich knihovnu, i když nechcete použit nic jiného než distribuci aplikací. Většina služeb neobsahuje historií nahraných verzí a možnost jejich stáhnutí.

Poslední nevýhoda je ta, že žádná nepodporuje brandování svých iOS aplikací. Vždy je lepší, když svému zákazníkovi pošleme aplikaci se svým logem a barvami, než s logem a texty úplně jiné společnosti.

Jelikož vyvíjený systém je řešení pro konkretní firmu, tak je výhodnější si postavit vlastní řešení, než použít existující. Důvodů je mnoho a hlavní z nich je, že hotové řešení půjde k zákazníkům, kde chceme, aby měli naši aplikaci a nepoužívali služby třetích stran. Při použití vlastního řešení jsme schopni reagovat na speciální požadavky a potřeby našich zákazníků a podle toho upravit

náš systém. Vlastní systém bude působit v očích zákazníka lépe ve prospěch naši firmy.

Služba	Registrace	iOS App	macOS App	CI	SDK	Historie	Cena
TestFlight	ANO	ANO	ANO	ANO	NE	NE	P
HockeyApp	ANO	ANO	ANO	NE	ANO	NE	FM
Crashlytics	ANO	ANO	ANO	ANO	ANO	NE	F
Supersend	NE	NE	NE	NE	NE	NE	F
TestFairy	ANO	ANO	NE	NE	ANO	NE	F
Appaloosa	ANO	ANO	NE	ANO	ANO	NE	FM
AppBlade	ANO	ANO	NE	ANO	ANO	NE	FM
BirdFlight	ANO	NE	NE	NE	NE	NE	F
Installr	NE	NE	NE	ANO	NE	NE	FM

Table 1: Shrnutí existujících služeb

Vysvětlivky k tabulce:
Sloupec *Registrace* - Znamená, jestli vyžadována registrace pro uživatele do služby.

Sloupec *iOS App* - Jestli služba poskytuje aplikaci pro iOS.

Sloupec *macOS App* - Jestli služba poskytuje aplikaci pro macOS.

Sloupec *CI* (**C**ontinous **I**ntegration) - Jestli služba nabízí možnost automatizovat nahrání nové verze.

Sloupec *SDK* - Znamená, jestli je nutné zahrnout do projektu SDK od služby, při použití pouze distribuce verzí.

Sloupec *Historie* - Jestli služba nabízí historií nahraných verzí a možnost zpětné instalace.

Sloupec *Cena* obsahuje tři hodnoty: **P**, **F**, **FM**, kde:
Paid - znamená, že je služba pouze placená
Free - znamená, že je celá služba zdarma
FM (**F**ree**m**ium) - znamená, že služba je zdarma s omezením, jinak je placená

4 Analýza řešení

Celý systém se skládá ze tří hlavních komponent, a to: Server, iOS Aplikace a macOS Aplikace. Server slouží ke správě uživatelů. macOS Aplikace slouží ke správě projektů a verzí. A poslední iOS Aplikace slouží k instalací aplikací na koncové zařízení.

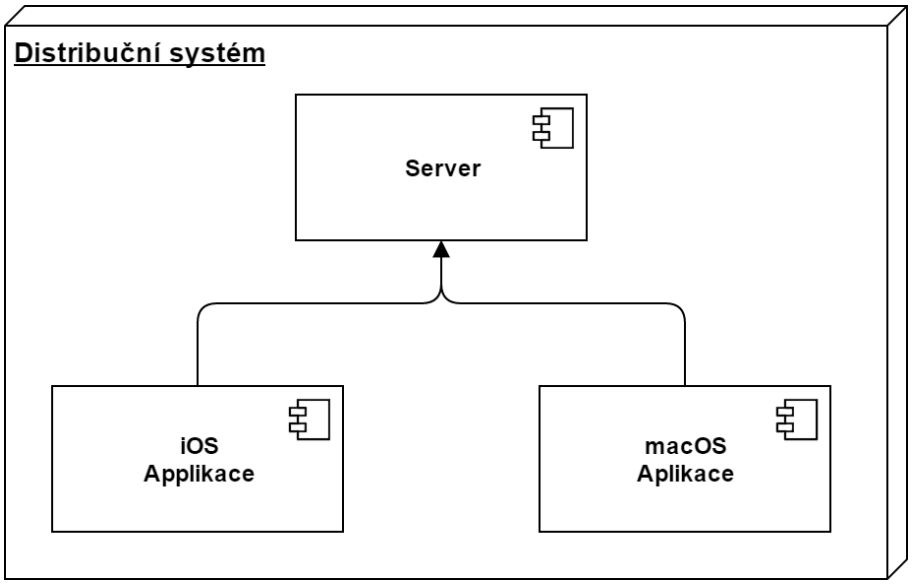


Figure 1: Distribuční systém - Komponenty

4.1 Případy užití

Případy užití jsou znázorněny pomocí UML diagramu. Obsahují obecné případy užití, které uživatel může s aplikacemi provádět. Obecné případy užití nebudou rozepsány do textové podoby. Složitější a zajímavější případy užití budou rozepsány.

4.1.1 iOS Aplikace

Obecné případy užití v iOS aplikaci.

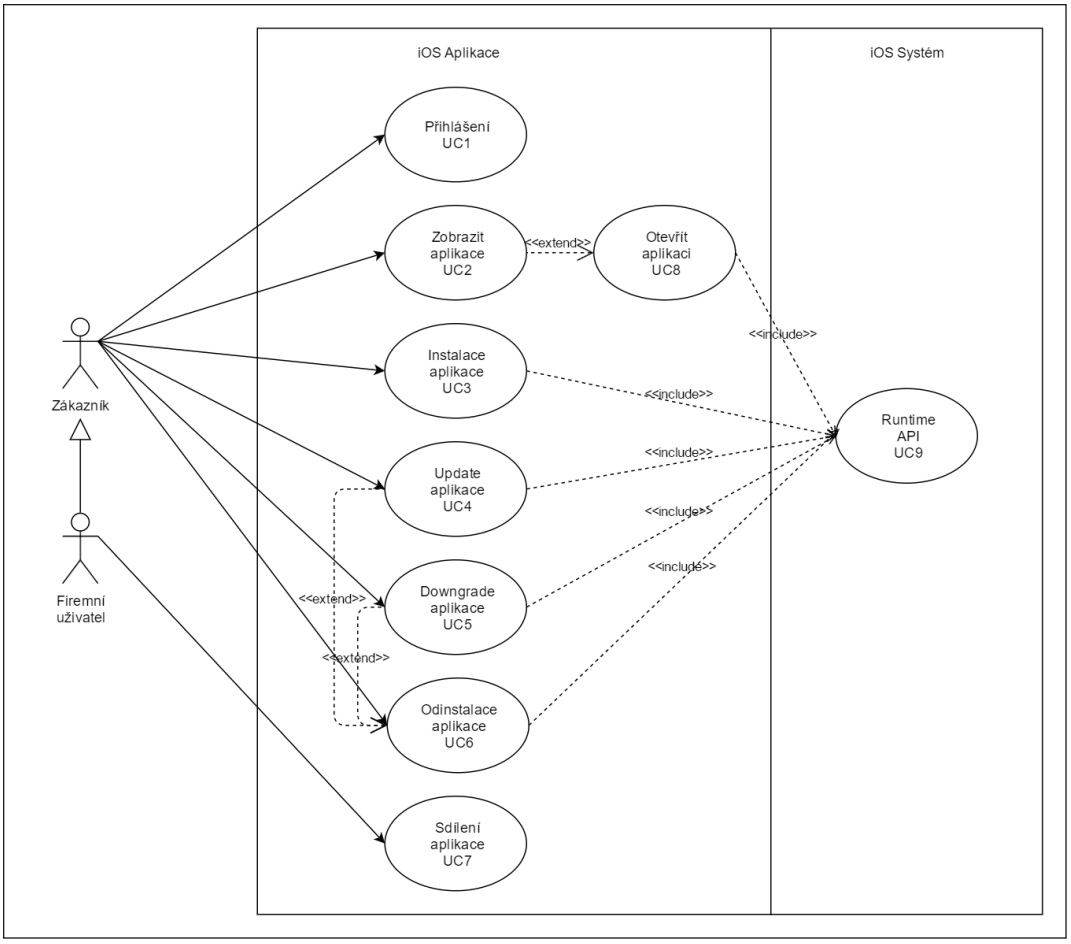


Figure 2: iOS případy užití

UC5 - Downgrade aplikace

Název	Downgrade aplikace
ID	UC5
Popis	Nainstaluje se starší aplikace s možností volby odinstalace aktuální aplikace.
Aktéři	Uživatel
Parametry	Vybraná verze k downgrade
Vstupní podmínky	Uživatel je přihlášen. Uživatel vybral verzi k downgrade
Výstupní podmínky	Uživatel má nainstalovanou starší verzi aplikace

Table 2: UC5 - Downgrade aplikace - popis

UC5 - Downgrade aplikace

Hlavní tok		
Krok	**Role**	**Akce**
1	**Uživatel**	Uživatel zvolí aplikaci k downgrade
2	**Aplikace**	Aplikace zobrazí dialog o upozornění, že může dojít k poškození dat, či ztrátě funkcionality. Jsou zobrazeny dvě tlačítka: ”Ano, pokračovat”, ”Nepokračovat”
3	**Uživatel**	Uživatel zvolí odpověď v dialogu
4	**Aplikace**	Aplikace zobrazí dialog, jestli si uživatel přeje odinstalovat aplikaci. Jsou zobrazeny dvě tlačítka: ”Ano, odinstalovat”, ”Neodinstalovat”
5	**Uživatel**	Uživatel zvolí odpověď v dialogu
6	**Aplikace**	Aplikace předá systému identifikátor aplikace k odinstalaci
7	Systém	Systém odinstaluje aplikaci
8	Aplikace	Aplikace předa systému odkaz na nainstalování aplikace
9	Systém	Systém nainstaluje aplikaci
10	Aplikace	Aplikace zobrazí uživateli výsledek
Vedlejší tok		
Krok	**Role**	**Akce**
3a	**Uživatel**	Jestli uživatel vybral nepokračovat, přeskakuje se ke kroku 10
4a	**Uživatel**	Jestli uživatel vybral neodinstalovat, přeskakuje se ke kroku 8

Table 3: Případ užití - Downgrade aplikace

4.1.2 macOS Aplikace

Obecné případy užití v macOS aplikaci.

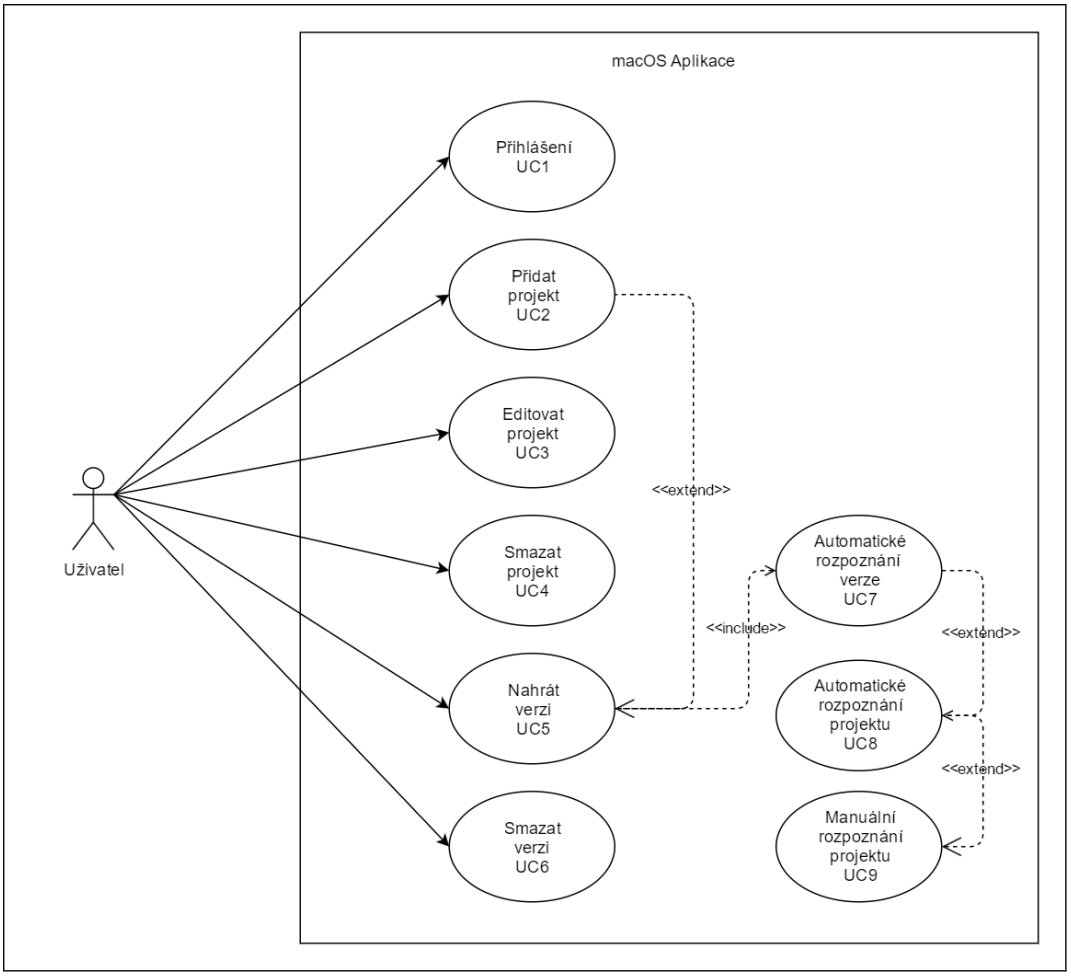


Figure 3: macOS případy užití

UC5 - Nahrát verzi

Název	Nahrát verzi
ID	UC5
Popis	Uživatel vybere soubor, který chce nahrát a přejde se k automatickému rozpoznání verze a projektu s možností manuálního výběru projektu.
Aktéři	Uživatel
Parametry	Soubor
Vstupní podmínky	Uživatel je přihlášen. Uživatel vybral soubor, který chce nahrát.
Výstupní podmínky	Uživatel nahrál do systému novou verzi.

Table 4: UC5 - Nahrát verzi - popis

UC5 - Nahrát verzi

Hlavní tok		
Krok	**Role**	**Akce**
1	**Uživatel**	Uživatel vybere soubor, který chce nahrát
2	**Aplikace**	Aplikace automaticky rozpozná verzi
3	**Aplikace**	Aplikace automaticky rozpozná projekt
4	**Aplikace**	Aplikace zobrazí uživateli informace o rozpoznané verzi a projektu
5	**Aplikace**	Aplikace umožní uživateli nahrát verzi
6	**Uživatel**	Uživatel nahraje verzi
Vedlejší tok		
Krok	**Role**	**Akce**
2a	**Aplikace**	Jestli nedojde k rozpoznání verze, tak scénář končí selháním a přeskakuje se na krok 1
3a	**Aplikace**	Jestli nedojde k rozpoznání projektu, tak je uživateli zobrazena možnost manuálního výběru projektu
3a1	**Uživatel**	Uživatel vybere projekt a přeskakuje se na krok 4

Table 5: Případ užití - Nahrát verzi

4.1.3 Webové rozhraní

Obecné případy užití ve webové aplikaci.

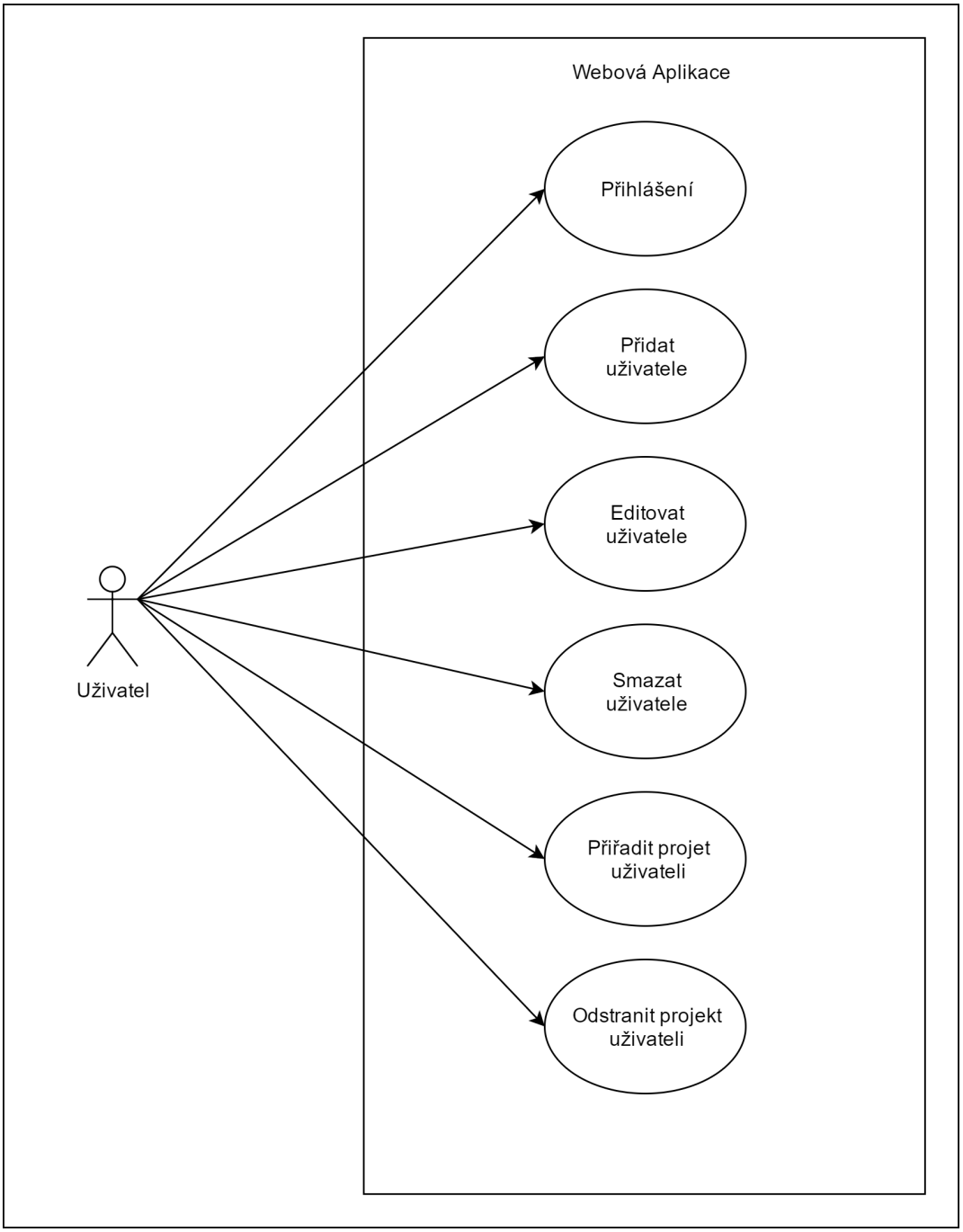


Figure 4: Webové případy užití

4.2 Požadavky

Podle analýzy konkurenčních aplikací a požadavků z praxe, jsou sestaveny následující požadavky.

4.2.1 Funkční požadavky

iOS Aplikace

1. **Jednoduché přihlášení pomocí odkazu**

Priorita: Hlavní
Popis: Uživateli je zaslán email obsahující odkaz, na který klikne v iOS systému a následuje otevření aplikace a přihlášení uživatele.

2. **Instalace aplikací**
   Priorita: Hlavní
   Popis: Uživatel má možnost si nainstalovat dostupné aplikace.

3. **Update aplikací**
   Priorita: Hlavní
   Popis: Uživatel má možnost aktualizovat již nainstalovanou aplikaci.

4. **Downgrade aplikací**
   Priorita: Hlavní
   Popis: Uživatel má možnost si zobrazit starší aplikace a vybrat si libovolnou verzi a následně ji nainstalovat. Uživateli je nabídnuta možnost smazat aktuálně nainstalovanou aplikaci.

5. **Smazání aplikací**
   Priorita: Vedlejší
   Popis: Uživatel má možnost si smazat libovolnou nainstalovanou aplikaci.

6. **Otevírání aplikací**
   Priorita: Vedlejší
   Popis: Uživatel má možnost si otevřít libovolnou nainstalovanou aplikaci.

7. **Zobrazení všech nainstalovaných aplikací**
   Priorita: Vedlejší
   Popis: Uživatel vidí všechny aplikace, které si nainstaloval.

8. **Zobrazení všech dostupných updatů**
   Priorita: Vedlejší
   Popis: Uživatel vidí všechny dostupné aktualizace pro aplikace, které má nainstalované.

macOS Aplikace

1. **Vytvoření projektů**
   Priorita: Hlavní
   Popis: Uživatel má možnost vytvořit nový projekt a přidat ho do systému. Po vytvoření je možnost nahrávat verze do projektu.

2. **Různé způsoby mazání projektů**
   Priorita: Vedlejší
   Popis: Aplikace nabízí tři možnosti mazání projektů, z existujícího stromu projektů.

3. **Nahrání nové verze**
   Priorita: Hlavní
   Popis: Uživatel má možnost nahrát novou verzi k projektu.

4. **Automatické rozpoznání projektu z vybrané verze**
   Priorita: Hlavní
   Popis: Při zvolení souboru nastane automatické rozpoznání verze a rozpoznání projektu s možností manuálního výběru projektu.

Server Aplikace

1. **Správa uživatelů**
   Priorita: Hlavní
   Popis: Slouží k přidání, odebrání a editaci uživatelů a přiřazení projektů k uživatelům.

2. **Zprostředkování dat přes REST API**
   Priorita: Hlavní
   Popis: Server nabízí data a služby prostřednictvím RESP API, které využívají ostatní aplikace k získání dat a ověření uživatelů.

3. **Možnost napojení na Continous Integration**
   Priorita: Vedlejší
   Popis: Možnost napojení systému na automatizační nástroje, jako například: build server.

4.2.2 Nefunkční požadavky

1. **iOS Aplikace vyžaduje minimální verzí 8.0**
   Priorita: Vedlejší
   Popis: Minimální verze iOS systému, na kterém je aplikace dostupná a podporována.

2. **macOS Aplikace vyžaduje minimální verzí 10.10**
   Priorita: Vedlejší
   Popis: Minimální verze macOS systému, na kterém je aplikace dostupná a podporována.

3. **Aplikace vyžaduje internetové připojení**
   Priorita: Vedlejší
   Popis: Aplikace nefungují bez internetového připojení a k použití kterékoliv části aplikace je potřeba být vždy připojen.

4. **Uživatelské rozhraní aplikací dodržuje standardy dané platformy**
   Priorita: Hlavní
   Popis: Každá platforma má svoje specifické chování a standardy, které je potřeba dodržet, aby se uživatel snadno vyznal v aplikacích.

5. **Přehledné uživatelské rozhraní**
   Priorita: Hlavní
   Popis: Pro uživatelé je potřeba jednoduché a přehledné uživatelské rozhraní tak, aby vše bylo dostupné a srozumitelné.

5 Návrh

5.1 Uživatelské rozhraní

Předtím, než se začne s reálnou implementací aplikací na obou platformách, je potřeba navrhnout drátěný model, neboli tzv. Wireframes. Drátěný model nám slouží k navrhnutí uživatelského rozhraní ještě předtím, než je skutečně implementovaný. Díky tomu můžeme najít nedostatky v chování, nebo v ovládání aplikace a odstranit je už na začátku.
Nástrojů pro navrhování drátěného modelu je v dnešní době hned několik. Některé nástroje jsou například určené pouze pro návrh webu, nebo mobilních systémů a díky tomu obsahují předem připravené grafické komponenty. Samotný návrh může obsahovat i chování aplikace jako například přechody mezi obrazovkami, animace a nebo dynamický obsah. V našem návrhu si vystačíme pouze se statickým návrhem uživatelského rozhraní pro všechny platformy.

5.1.1 iOS

Na návrh drátěného modelu pro platformu iOS byl použit software Justinmind. Díky tomu, že nástroj podporuje platformy iOS, Android a Web, tak obsahuje pro ně už existující grafické komponenty, které jsou standardem pro danou platformu a není je potřeba vytvářet znova, ale stačí je pouze použít a tím urychlit návrh drátěného modelu.
Návrh aplikace dodržuje standardy doporučované pro uživatelské rozhraní a chování na iOS platformě.

Jako základní navigace v aplikaci je TabBar, který obsahuje tři záložky. První záložka je Projects, která zobrazuje informace o dostupných projektech. Druhá záložka je Apps, která zobrazuje aktuálně nainstalované aplikace. A poslední záložka je Updates, která zobrazuje dostupné updaty pro aplikace.

Popíšu zde ty nejdůležitější a nejzajímavější obrazovky z aplikace. Popis všech ostatních obrazovek je v příloze.

Projects
Obrazovka se stará o zobrazení všech dostupných projektů. Má dva druhy grafického rozvržení, které si uživatel může sám přepínat, podle toho jak mu to vyhovuje. První rozvržení je tabulka obsahující řádky přes celou šířku obrazovky s názvem projektu. Druhé rozvržení je zobrazení projektů do mřížky s ikonkami a názvem. Dále se zde nachází vyhledávací pole pro rychlejší orientaci. Vyhledávací pole je ve výchozím stavu schované a je potřeba provést akci nazývanou Swipe to Down pro zobrazení a použití. Klikem na ikonku nebo řádek tabulky se dostáváme buď na detail projektu a nebo na zobrazení subprojektů vybraného projektu. Obrázek 5 a 6

Projects
Hlavní obrazovka projektu, kde jsou vidět jeho detaily a možnost instalace různých verzí aplikace, respektive projektu.

Ve vrchní části obrazovky jsou základní informace o projektu, které jsou název a ikonka aplikace. Pak se zde nachází tabulka obsahující dostupné verze aplikace, které si uživatel může nainstalovat. Seznam je řazený podle poslední aktivity v dané verzi. Uživatel má možnost u každé verze provádět akce, podle toho v jakém stavu se nachází daná verze. Stavy mohou být: Install, Open, Update a Downgrade. Install stav je zobrazen když uživatel ještě nemá aplikaci nainstalovanou na svém zařízení. Po nainstalování se aplikace dostává do stavu Open. Pak podle toho, jestli je dostupná nová verze, tak je zobrazený stav Update a v opačném případě, kdy uživatel si zobrazuje starší verze aplikace je zobrazena možnost Downgrade. Uživatel u každé verze vždy vidí její číslo a datum vytvoření.

Někdy při testování aplikací je potřeba si nainstalovat předchozí verze aplikace, ať už z důvodu otestování přechodu ze starší verze na novou, nebo jakéhokoliv jiného. Tahle obrazovka v základu zobrazuje pouze nejnovější verze aplikace, jelikož to je pro největší počet uživatelů to nejdůležitější. Nicméně nabízí možnost zobrazit i předchozí verze a to pomocí tažením prstu z dolů směrem nahoru. Při dostatečném zatažení dojde ke stažení starších verzí a následném zobrazení uživateli. Obrázek 7

Installation
Obrazovka se otevírá po kliku uživatelem na nějakou verzi aplikace v určitém stavu a to: Install, Update a Downgrade. Jedná se obrazovku, která zobrazuje postup při instalaci aplikace. Uživatel první povolí, že chce aplikaci nainstalovat a potom dochází k samotné instalaci, která se zobrazuje graficky, aby uživatel věděl co se děje. Instalace může skončit dvěma stavy a to: Uspěla a Neuspěla. Obrázek 8

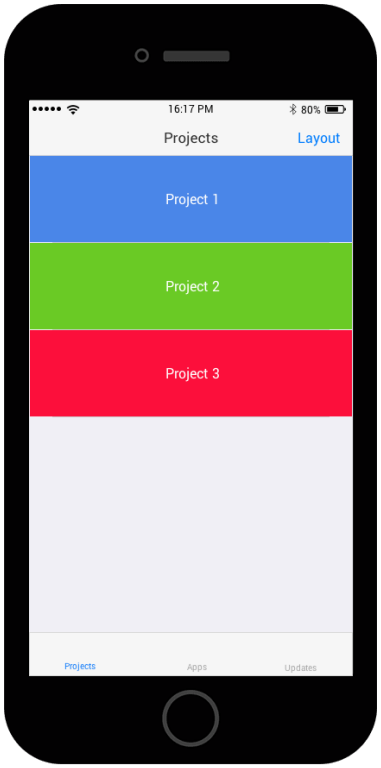


Figure 5: Obrazovka - Projects, rozvržení do tabulky

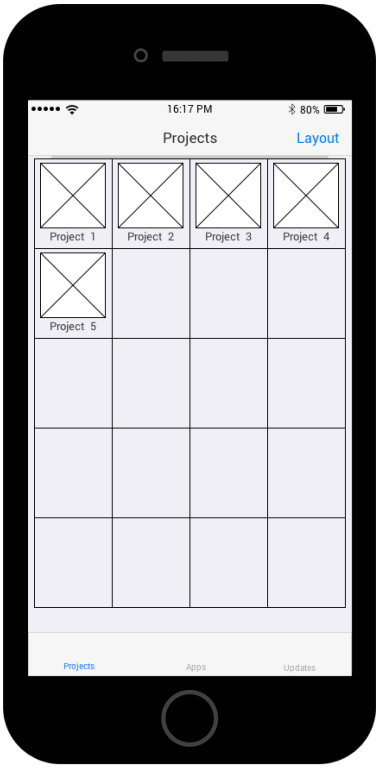


Figure 6: Obrazovka - Projects, rozvržení do mřížky

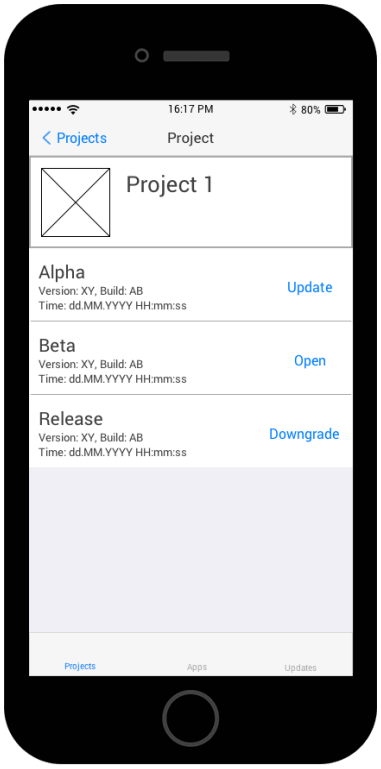


Figure 7: Obrazovka - Project Detail

Figure 8: Obrazovka - Installation

Apps

Jedná se o obrazovku, která zobrazuje v mřížce už nainstalované aplikace. Uživatel má tedy možnost ihned si vybrat jakoukoliv aplikaci a spustit ji. Není potřeba hlavní aplikaci minimalizovat a hledat své nainstalované aplikace. Stejně jako obrazovka Projects, tak i tato obsahuje funkcionalitu Swipe to Down pro zobrazení vyhledávacího pole pro rychlejší vyhledání požadováné aplikace. Obrázek 9

Updates

Zde je uživatel informován o všech dostupných updatech aplikací, které má nainstalované. Tabulka s aplikacemi seskupuje všechny dostupné updaty podle projektů a v každém seskupení se nachází všechny dostupné verze, pro které jsou updaty. Obrázek 10

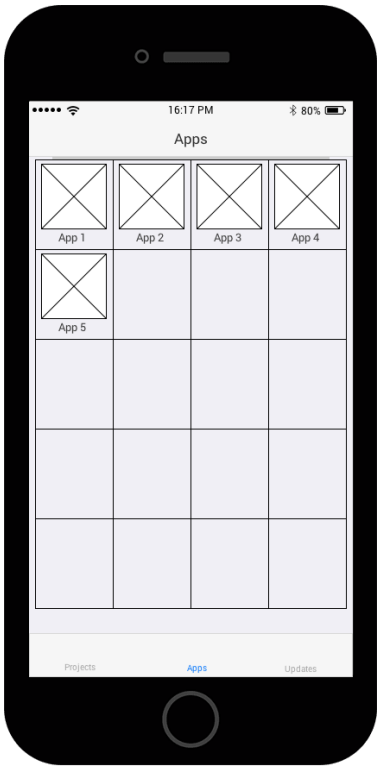


Figure 9: Obrazovka - Apps

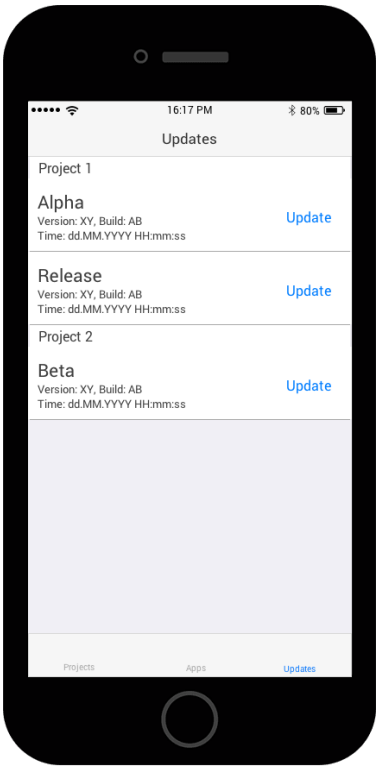


Figure 10: Obrazovka - Updates

5.1.2 macOS

Na návrh drátěného modelu pro platformu macOS byl použit software Axure. Návrh aplikace dodržuje standardy doporučované pro uživatelské rozhraní a chování na macOS platformě.

Jako základní navigace v aplikaci je horní tab bar, který obsahuje šest záložek, které jsou rozděleny do tří logických skupin.

První skupina se zabývá správou projektů.

První záložka je Add Project, která umožňuje založení nového projektu. Druhá záložka je Edit Project, která dovoluje aktualizovat již existující projekt. A poslední záložka je Delete Project, která umožňuje různé způsoby odstranění projektů.

Druhá skupina se zabývá správou verzí.
První záložka je Add Builds, která umožňuje nahrát novou verzi do konkretního projektu. Druhá záložka je Delete Builds, kde má uživatel možnost smazat jakoukoliv verzi z projektu.

Poslední skupina obsahuje ostatní akce.
Nachází se zde jedno tlačítko, které slouží k odhlášení existujícího uživatele.

Popíšu zde ty nejdůležitější a nejzajímavější obrazovky z aplikace. Popis všech ostatních obrazovek je v příloze.

Obrazovka - Add Project
Jedna z hlavních obrazovek. Je zde možnost vytvořit nový projekt. Při vytvoření projektu musí být vyplněny základní atributy. Povinné atributy jsou: jméno, barva textu a barva pozadí. Nepovinné atributy jsou: App Store ID, rodičovský projekt a možnost nahrát verzi do projektu při vytváření projektu. Barva textu a barva pozadí se používá v iOS aplikaci v obrazovce Projects v rozvržení do tabulky.

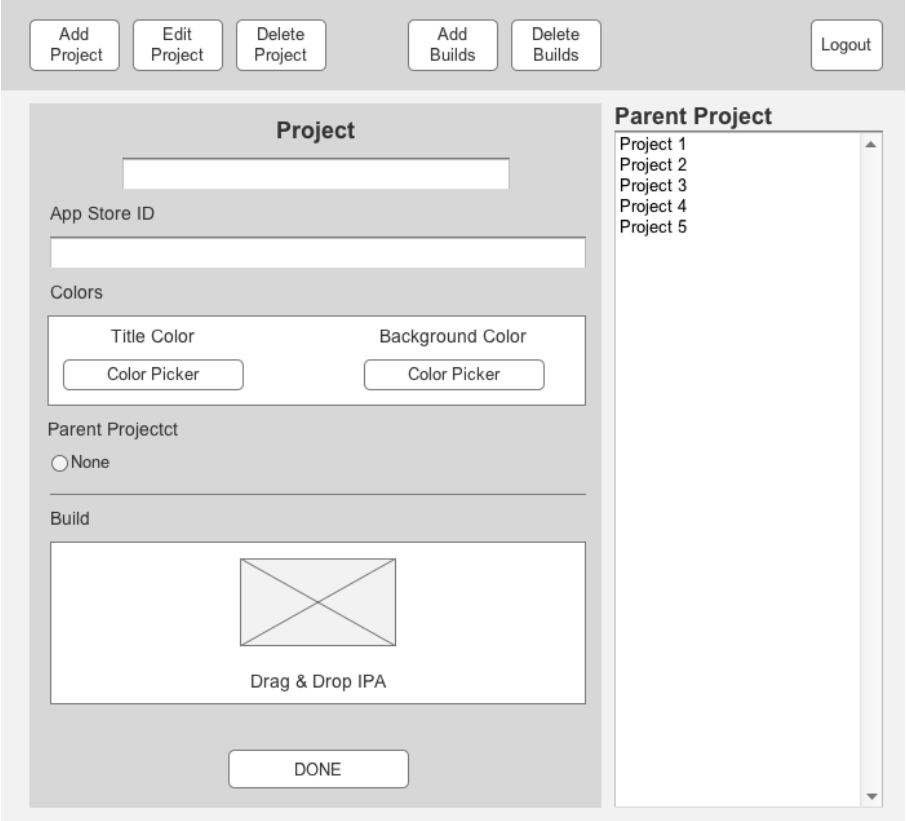


Figure 11: Obrazovka - Add Project

Obrazovka - Delete Project

Tahle obrazovka slouží k odstraní projektů ze systému. Jsou zde tři možnosti odstranění. První je References, druhá je Project a poslední možnost je Recursively. Podrobný popis, jak která funkce funguje je rozebrán v Analýze v sekci Architektura.

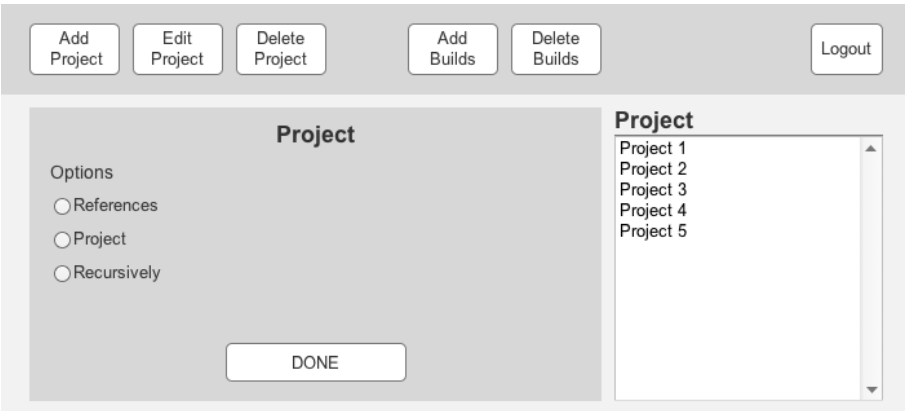


Figure 12: Obrazovka - Delete Project

Obrazovka - Add Build

Druhá hlavní obrazovka, která umožňuje nahrát verzi do projektu. Obrazovka je velmi jednoduchá, obsahuje pouze jedno Drag and Drop pole pro vybrání souboru, který se má nahrát. Jestliže projekty už obsahují nahrané verze, tak dojde k automatickému rozpoznání projektu podle vybraného souboru a uživatel tak nemusí vybírat manuálně, do kterého projektu chce novou verzi nahrát. Dojde taky k rozpoznání o jakou verzi se jedná, jestli Alpha, Beta nebo Release. Podrobný popis, jak rozpoznání funguje je rozebrán v Analýze v sekci Architektura.

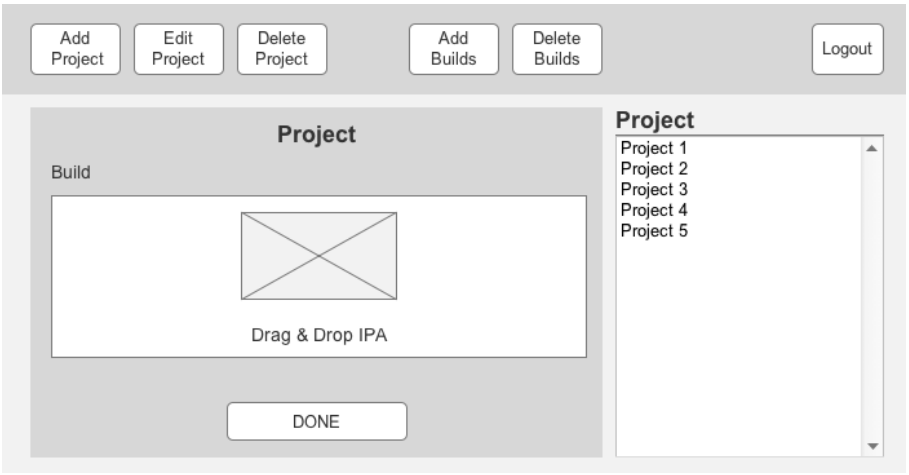


Figure 13: Obrazovka - Add Build

5.1.3 Web

Správa uživatelů v systému probíhá přes webové rozhraní. Uživatelské rozhraní pro web používá klasický design Django. Základní Django rozhraní je dostačující pro dané požadavky.

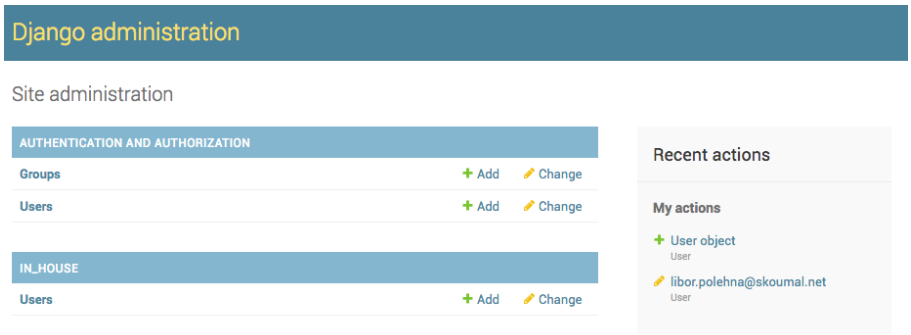


Figure 14: Web - Správa uživatelů

5.2 Architektura

Návrh architektury systému můžeme rozdělit podle platformy. Zatímco na serveru je použita MVT architektura, tak na platformách iOS a macOS je MVC architektura. Celý systém, jak server, tak aplikace pro iOS a macOS, můžeme rozdělit taktéž na tři vrstvy. První vrstva slouží pro ukládání dat. Druhá vrstva slouží pro komunikaci se serverem. A poslední vrstva je samotná logika jednotlivých aplikací. Dále si jednotlivě rozebereme důvody, proč byly pro vývoj použité tyhle architektury.

5.2.1 Server - MVT architektura

Volba architektury se na serveru odvíjí od použitého frameworku. Použitý framework je Django z důvodu rychlého vývoje a podpory. Proto byla použita architektura MVT, neboli Model - View - Template. [10]
Architektura se skládá ze tří základních vrstev. Vrstva model slouží pro ukládání dat. Další vrstva View slouží pro byznys logiku, v MVT architektuře označované jako View, kde můžeme nalézt logiku aplikace, zpracování dat, výpočty atd. Poslední vrstva je prezentační vrstva. Ta slouží pro zobrazení informací uživatelům, v MVT označované jako Template.
Při implementaci se z architektury použili pouze dvě části, a to: Model a View. V modelu jsou definované struktury pro ukládání dat. A view obsahuje zpracování požadavků z REST API. Takže architekturu MVT můžeme zúžit na dvojvrstvou architekturu Model a View.
Celou architekturu obstarávají na serveru dva frameworky. První z nich je Django, který se stará o ukládání dat, logiku a grafický přistup do systému. Druhý framework je Django REST Framework a ten se stará o zpřístupnění dat ze systému jiným aplikacím.
Django REST Framework usnadňuje při vývoji REST API spoustu práce. Obsahuje předpřipravené funkce jako například: mapování objektů, specifikaci API, serializaci a deserializaci dat, autorizaci a autentizaci uživatelů a mnoho dalšího.
Pro výměnu dat mezi aplikacemi a serverem byl vybrán formát JSON. Jedná se o velmi jednoduchý a rozšířený formát. Na platformách iOS a macOS se používá nejčastěji a podpora pro něj je velmi dobře implementována v systémových knihovnách, na rozdíl od XML, a právě proto byl vybrán a použit. Za zvážení stojí ještě Proto Buffer od Google. Je to mechanizmus na serializaci dat. Je nezávislí na platformě, je rychlejší a úspornější než JSON a XML. Přesto má i nějaké nevýhody. Jedná se o binární formát, takže není lidsky čitelný. Dále Google neposkytuje podporu pro všechny programovací jazyky, takže například Swift od Apple není Googlem podporován, ale komunitou. I přes všechny výhody Proto Bufferů oproti JSON formátu zůstal vybrán JSON. Hlavní důvod je kvůli zachování kompatibility s co nejvíce platformy, které mohou být v průběhu používání systému přidány.

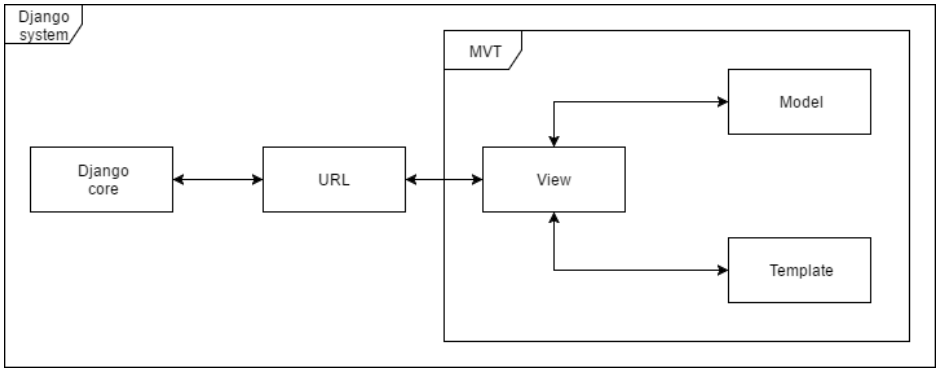


Figure 15: MVT & Django architektura

5.2.2 iOS a macOS - Apple MVC architektura

Při hledání vhodné architektury se nabízelo několik možností. Hlavní z nich jsou MVVM, MVC a Viper [11, 12, 13]. Nejdůležitější bylo najít architekturu, která bude vyhovovat požadavkům na vyvíjený systém. Jelikož systém není velkého rozsahu, tak Viper architektura nevyhovovala potřebám a proto nebyla použita. Z posledních dvou architektur byla vybrána MVC. A to z důvodu jednoduchosti použití a implementace.

Nejedná se o klasickou MVC architekturu, ale mírně upravenou firmou Apple. Architektura se také skládá z hlavních tří bloků. Je zde model, který se stará o data. View, které se stará o zobrazení informací uživateli. Poslední blok je controller, který obstarává komunikaci mezi modelem a view.

Komunikace mezi bloky je následující. Controller zasílá události do modelu a model notifikuje controller o změně dat. Dále controller aktualizuje view, když je potřeba, a zároveň view posílá události controlleru, když uživatel provede nějakou akci.

Při použití MVC architektury zůstane model a view od sebe odstíněné a tak na sobě nejsou závislé. Veškerou komunikaci mezi nimi obstarává controller. Jako každá architektura má svoje výhody i nevýhody. U MVC architektury se objevují její problémy až při implementaci většího systému, kde controllery obsahují spoustu logiky (například logika pro transformaci dat, které budou zobrazeny uživateli) a architektura se stává s přibývající funkcionalitou čím dál tím méně přehledná, špatně udržovatelná a složitě rozšiřitelná.

I přes nedostatky MVC byla architektura použita při vývoji. Jelikož vyvíjený systém není příliš velký, ale spíše menšího rozsahu, tak architektura je plně dostačující. Další důvod použití MVC je ten, že iOS a macOS systémové frameworky jsou postaveny na MVC architektuře.

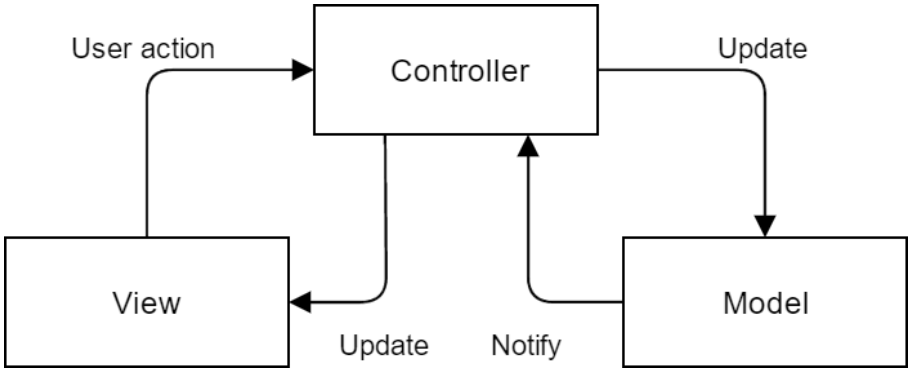


Figure 16: MVC architektura

5.2.3 Doménový model

Schéma databáze je velmi jednoduché. Obsahuje pár entit, které jsou spolu provázané.

Project
Obsahuje informace o projektu. Jako je: jméno, barva pozadí, barva textu, app store identifikátor a odkaz na rodičovský projekt. Projekt má vazbu sám na sebe, takže vzniká stromová struktura. Projekt může mít pouze jednoho rodiče a N potomků.

Build
Slouží k uchování informací o nahrané verzi. Obsahuje: identifikátor aplikace, build verzi, build číslo, typ, odkazy na soubory, datum vytvoření a odkaz na projekt do kterého patří.

User
Hlavní entita obsahující informace o uživateli. Například: jméno, email, heslo, kód, jestli je to zákazník a nebo uživatel z vlastní firmy. Uživatel má přiděleno několik projektů, se kterými může pracovat. Jestliže uživatel je z vlastní firmy, nemá přidělené žádné projekty, ale dochází k zpřístupnění všech projektů.

Login
Jedná se o entitu ukládající veškeré přihlášení od uživatele ze zařízení. Token se kterým zařízení komunikuje se serverem nemá expiraci, takže nedochází ke smazání jakéhokoliv záznamu z Login entity.

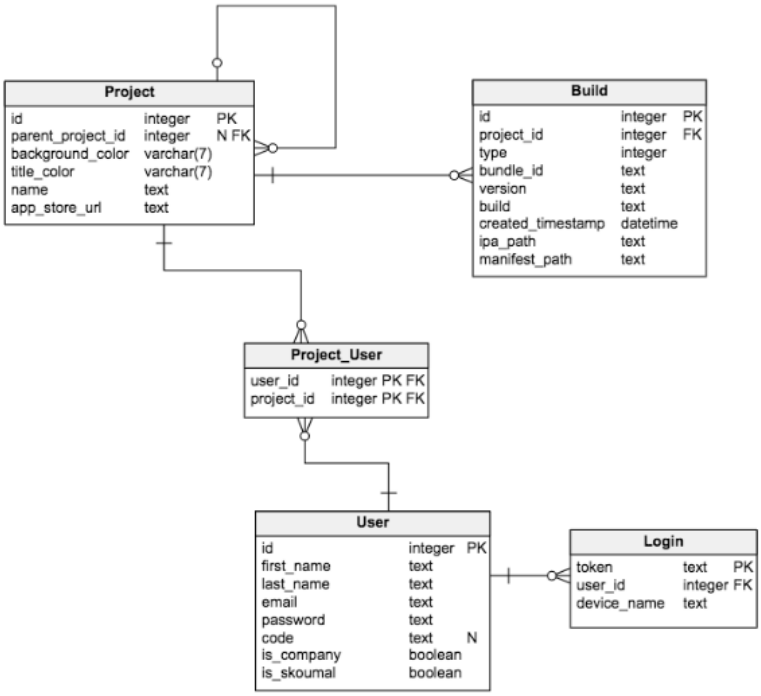


Figure 17: Schéma databáze

5.2.4 Struktura projektů

Každý projekt může mít jednoho rodiče a mnoho potomků. Jestliže projekt nemá rodiče, tak se jedná o kořenový projekt. Kořenových projektů může být neomezeně. Taky hloubka stromu je neomezená.

Na následujícím obrázku můžeme vidět příklad struktury projektů. Kde Project 1, Project 2 a Project 3 jsou kořenové projekty. Jestliže si představíme virtuální Project 0 nad prvními třemi projekty, tak nám vznikne stromová struktura s jedním kořenovým projektem. Strom by v praxi měl být pouze acyklického typu, ale vytvoření cyklu ve stromu není omezeno.

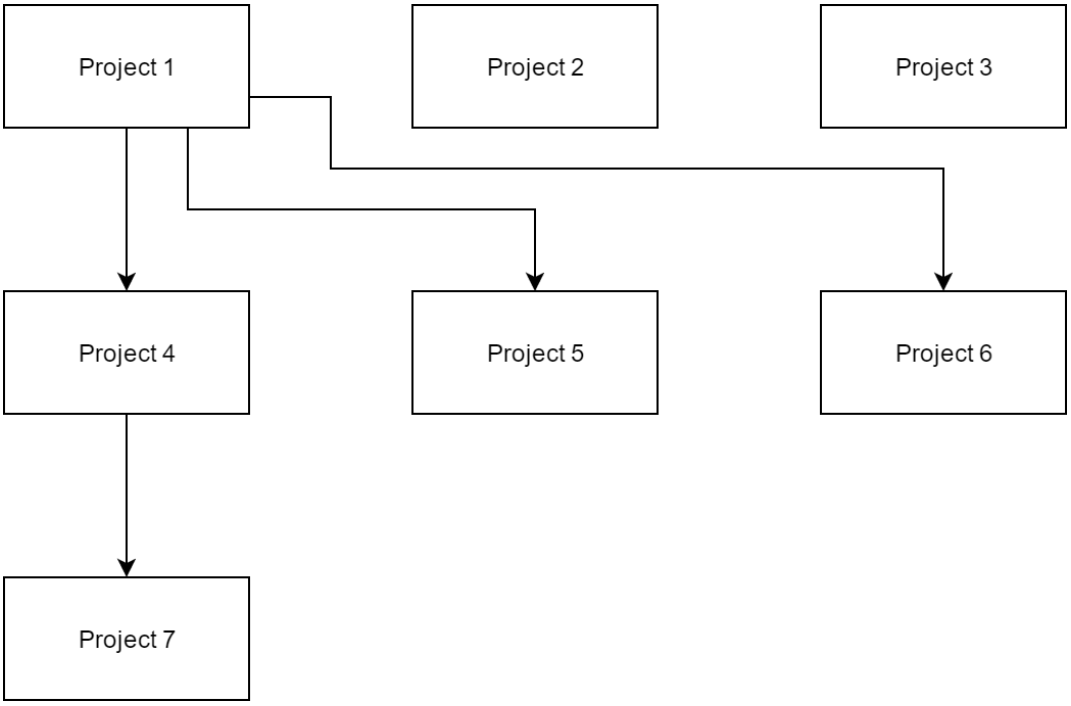


Figure 18: Příklad struktury projektů

Mazání projektů

Jelikož struktura projektů je stromová, tak se nabízí několik typů mazání projektů ze struktury. Ukázková struktura projektů, na které budou prezentovány metody mazání.

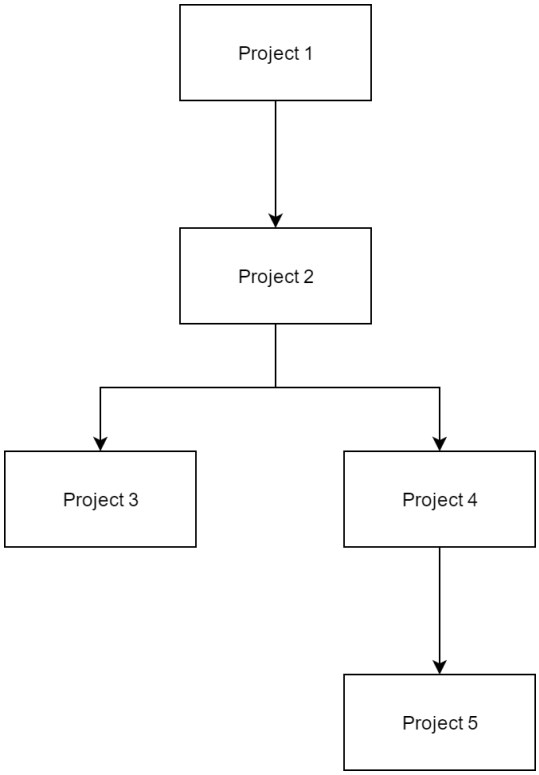


Figure 19: Příklad struktury projektů

První varianta je rekurzivní. Dojde ke smazání vybraného projektu a všech jeho potomků. Smažeme-li tedy Project 2 z ukázkového obrázku, tak nám zůstane pouze Project 1, který nebude mít žádné potomky.

Druhá varianta je smazání referencí na vybraný projekt. Smažeme-li znova Project 2, tak Project 1 nebude mít žádného potomka. Z projektů Project 2, Project 3 a Project 4 se stanou kořenové projekty.

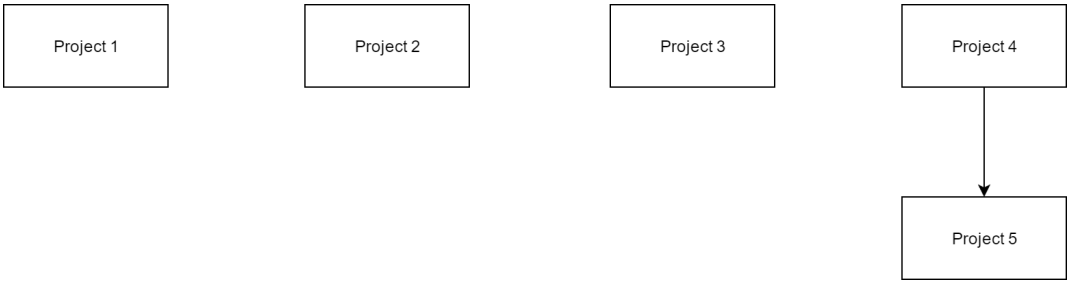


Figure 20: Smazání referencí

Poslední možností mazání je smazání samotného projektu, ale zachování jeho potomků. Smažeme-

li stejně Project 2, tak dojde k jeho odstranění a všechny jeho přímí potomci se stanou kořenovými projekty.

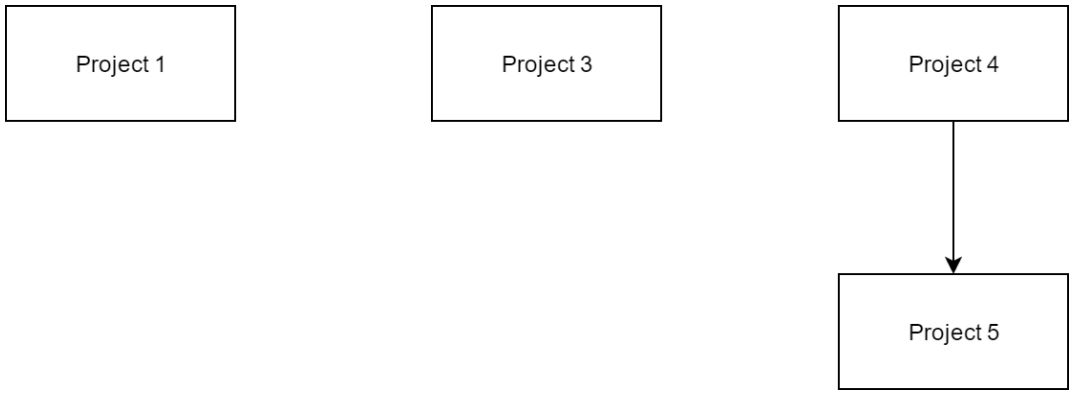


Figure 21: Smazání projektu

5.2.5 Diagram komponent

Celý systém lze rozložit do několika spolu komunikujících komponent.

Server
Komponenta obsahuje uložení dat, byznys logiku a přistup k datům, které nabízí klientům přes REST API.

Dynamic Framework
Zde je obsažena společná logika pro iOS a macOS aplikace. Jedná se převážně o komunikace se serverem, mapování dat z JSON na lokální objekty a základní byznys logika.

iOS Application
Komponenta, která obsahuje kompletní iOS aplikaci. Obsahuje uživatelské rozhraní a byznys logiku. Komponenta využívá další komponenty pro správný běh. Komponentu Dynamic Framework pro komunikaci se serverem a získání potřebných dat a komponentu iOS System pro komunikaci se systémem a využití jeho vestavěných frameworků a runtime API.

iOS System
Komponenta systému iOS, kde se kromě standardních frameworků využívá runtime API pro instalaci aplikací.

macOS Application
Zde se nachází pouze uživatelské rozhraní aplikace. Komponenta komunikuje s komponentou Dynamic Framework pro manipulaci s daty na serveru.

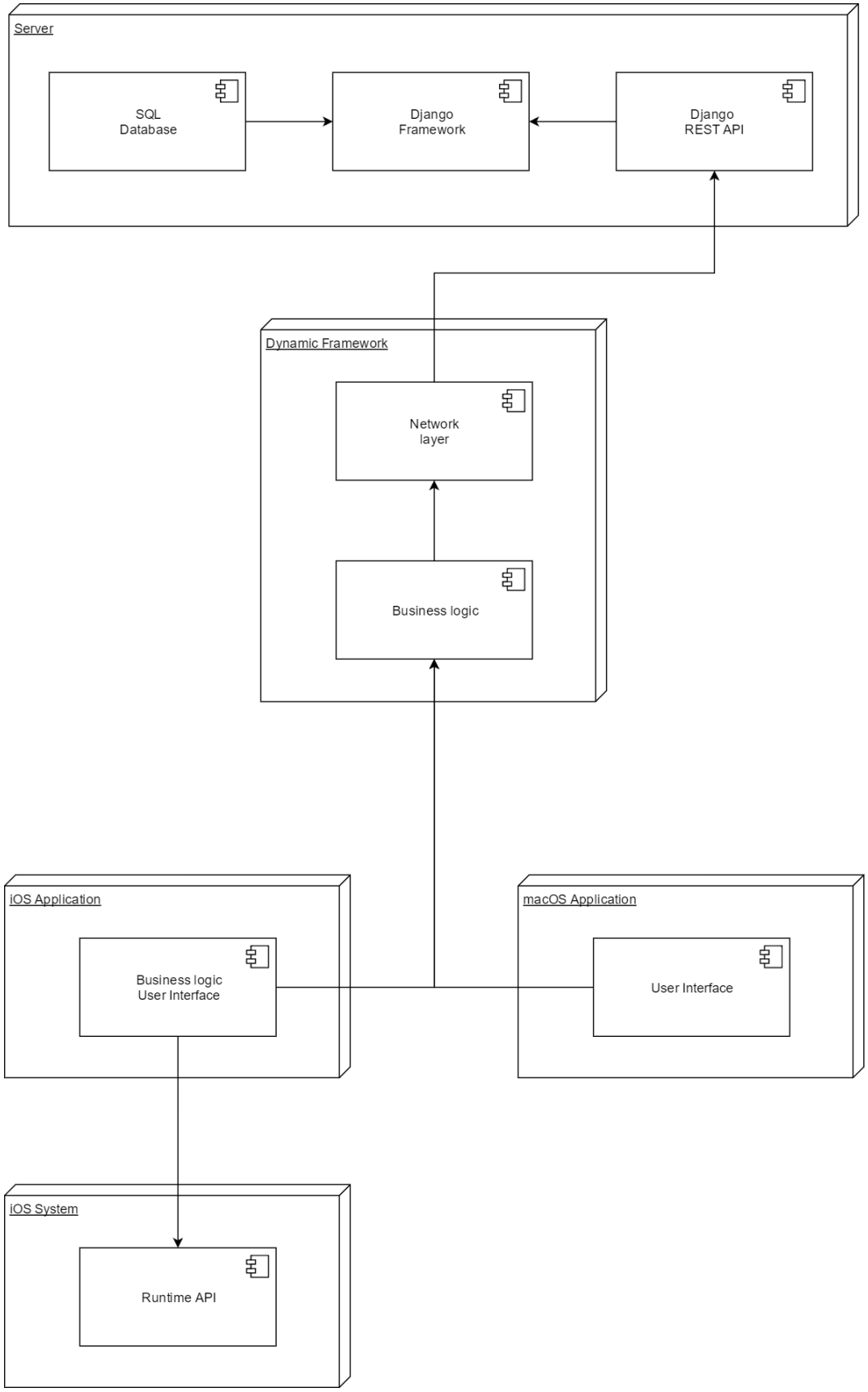


Figure 22: Diagram komponent

6 Implementace

Pro vývoj je důležité zvolit správné nástroje. Nástroje nám usnadňují celkový vývoj a zabraňuji zbytečným chybám. Celkově tak zrychlují a zkvalitňují vývoj.
Pro vývoj na straně serveru byl použít programovací jazyk Python. A pro vývoj aplikací pro iOS a macOS byly použity jazyky dva. Starší jazyk Objective-C a nový jazyk Swift. Objective-C bylo použito pouze pro podporu Runtime API.

6.0.1 Xcode

Vývoj iOS a macOS aplikací probíhal ve vývojovém prostředí Xcode [14]. Xcode pochází přímo od společnosti Apple. Jedná se o rozsáhle prostředí, které nabízí spoustu funkcí na zrychlení a zkvalitnění vývoje. K Xcode se také dodává sada simulátorů, které simulují reálné zařízení. Simulátory jsou dostupné pro každé zařízení, které Apple fyzicky prodává, čí prodával.

6.0.2 Carthage

Jedná se o nástroj, který zajišťuje správu závislosti na knihovnách od třetích stran. Udržuje je aktuální, hlídá závislosti mezi sebou a závislosti na verzích knihoven [15].

6.0.3 PyCharm

Nástroj je od společnosti JetBrains a slouží k vývoji v jazyce Python. Taktéž se jedná o komplexní vývojové prostředí, které nabízí nespočet funkcí pro snadnější vývoj a debugging Pythonu [16].

6.0.4 GIT

GIT je verzovací systém, který slouží pro správu změn při vývoji projektu. Jeho hlavní vlastnosti jsou například: podpora pro nelineární vývoj, distribuovaná vývoj, kryptografická autentizace historie, zjištění stavu projektu v čase a mnoho dalšího [17].

6.1 Serverový backend

Server zajišťuje hlavní logiku aplikace a samotné uložení a správu dat. Pro ukládání dat byla zvolena SQL databáze. Server je postaven na Django frameworku a Django REST frameworku.

6.1.1 Django

Django je open source moderní webový framework pro tvorbu webových stránek [18]. Je napsaný v jazyce Python a díky tomu podporuje rychlý vývoj služeb. Django dodržuje architekturu Model - View - Controller. Obsahuje spoustu už předem naimplementovaných funkcí jako například: ORM, administrační rozhraní, šablony, cachovací systém, lokalizaci, atp. Pro podporu implementace REST API na serveru existuje rozšiřující framework pro Django a to Django REST Framework. REST Framework usnadňuje práci při implementaci REST API a to díky

tomu, že obsahuje předem připravené nástroje jako jsou: ověření uživatelů, serializaci a deserializaci dat, generické views, validátory, oprávnění, stránkování, atp.

6.2 iOS a macOS

Při vytváření iOS a macOS aplikace je potřeba si velmi dobře navrhnout architekturu. Jelikož iOS a macOS jsou systémy od stejné společnosti, tak je možné některé části aplikací sdílet mezi sebou. Nejčastěji se sdílí logika a model aplikací. Ze společného kódu se vytvoří knihovna, která se do jednotlivých aplikací už pouze nalinkuje. Samotné aplikace v daném systému se starají především pouze o vytvoření uživatelského rozhraní. Logika a model systému je v knihovně a aplikace pouze využívá její služby.

Jsou dvě možnosti jak sdílet kód mezi aplikacemi. Můžeme použít statickou nebo dynamickou knihovnu. Hlavní rozdíly mezi typem knihoven jsou především výkon aplikací při startu a paměťová náročnost aplikace a v neposlední řade taky velikost výsledného spustitelného souboru. Na Apple systémech se dynamické knihovny často označují jako Dynamic Framework, proto dále budu používat toto názvosloví.

6.2.1 Dynamic Framework

Dynamický Framework, na rozdíl od statických knihoven, funguje odlišným způsobem. Zrychluje start aplikací, velikost spustitelného souboru a redukuje množství použité operační paměti aplikací [19].

Dynamiký Framework není při kompilaci vložen do výsledného spustitelného souboru. Načítá se a linkuje se do aplikace až v momentě, kdy je to potřeba. Samotný Dynamický Framework je umístěn v paměti typu Stack. Aplikace je umístěna v paměti typu Heap a má odkaz na framework do paměti Stack. Tím se šetří místo v paměti. Viz. obrázky [20, 21]

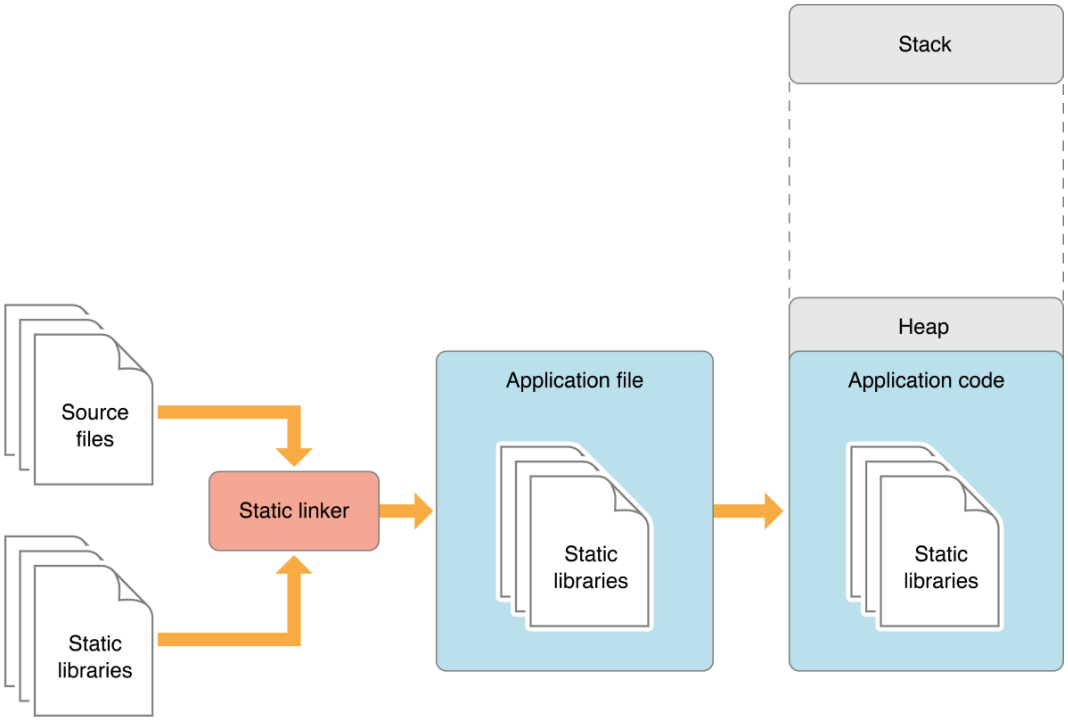


Figure 23: Statická knihovna

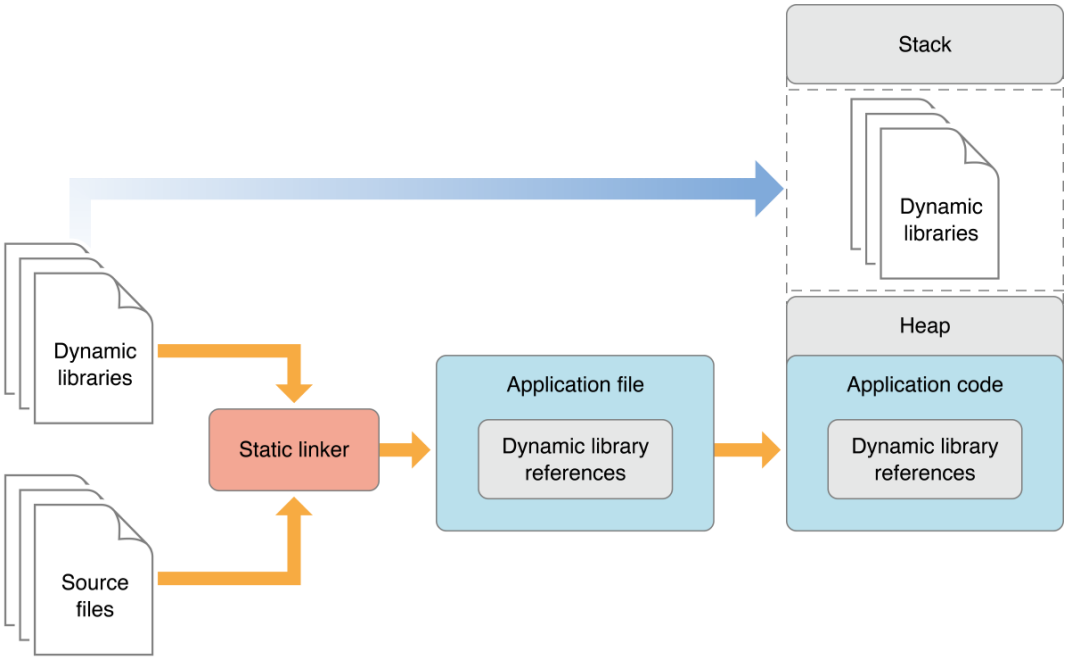


Figure 24: Dynamická knihovna

6.2.2 Použité knihovny

Mimo standardních knihoven se využívají knihovny od třetích stran. Většina knihoven pouze zaobalují systémové knihovny a zjednodušují tak jejich použití nebo je i rozšiřují vlastními algo-

ritmy. Ve výsledku se jedná o zrychlení vývoje aplikací. Při implementaci bylo použito několik knihoven od třetích stran.

AFNetworking
Jak už z názvu vypovídá, knihovna slouží pro práci se sítí [22]. Jedná se o jednu z nejrozšířenějších knihoven. Knihovna využívá systémové knihovny pro práci se sítí a zároveň přidává nové funkce. Knihovna je použita pro komunikaci se serverem a výměnu dat mezi sebou. Mimo jiné se využívá funkce na rozpoznání, zda je dostupné internetové připojení.

EasyMapping
Knihovna, která slouží k převodu JSON objektu na lokální objekty [23]. Knihovna obstarává mnoho funkcí automaticky, například validaci dat při převodu, nebo konverzi dat na jiné datové typy. Podobných knihoven je nespočet. Ta byla zvolena z důvodu, že umožňuje mapovat zanořené objekty. Zanoření používáme při parsování stuktury projektů.

Crashlytics
Když dojde k pádu aplikace na straně uživatele, tak bez fyzického přístupu k jeho zařízení není možnost, jak získat záznam o pádu a zjistit co se stalo a následně příčinu odstranit. K tomu existují služby, které se připojí k projektu a monitorují stav aplikace. Když dojde k pádu, tak služba eviduje pád a odešle detailní záznam na své servery, kde je ve webovém rozhraní vidět záznam o pádu. Následně jsme schopni příčinu odstranit a vydat aktualizaci aplikace.

6.3 iOS Aplikace

iOS Aplikace je hlavní části celého systému. Je to také nejsložitější implementační část, jelikož se zde snažíme obejít uzavřenost iOS systému a využití některých jeho části k dosažení námi definovaných potřeb. Obcházení systému probíhá pomocí Runtime API, takže si ho blíže rozebereme.

6.3.1 Runtime Introspection

Runtime, někdy taky označované jako "execution time", je to stav, ve kterém program běží, respektive je prováděn. Doba, za kterou se program považuje ve stavu runtime, je od doby kdy, je program spuštěn, až do doby, kdy je jeho běh ukončen. Při spuštění dochází k načtení programu do operační paměti a všech jeho součástí jako například: knihovny, frameworky, spustitelný soubor a ostatní soubory, které jsou odkazované programem. Jakmile dojde k ukončení programu, tedy ke konci stavu označované runtime, operační paměť, kterou program zabíral, je uvolněna a program skončil [24, 25].

Nachází se zde také Runtime Library. Runtime Library je sada běžných postupů (rutin) sloužící k vyvolání některého chování z runtime prostředí. Runtime prostředí implementuje zpracování výjimek, správu paměti a základní chování programovacího jazyka. Díky tomu je runtime library vždy specifická pro danou platformu.

Ukázka struktury Runtime v Objective-C.

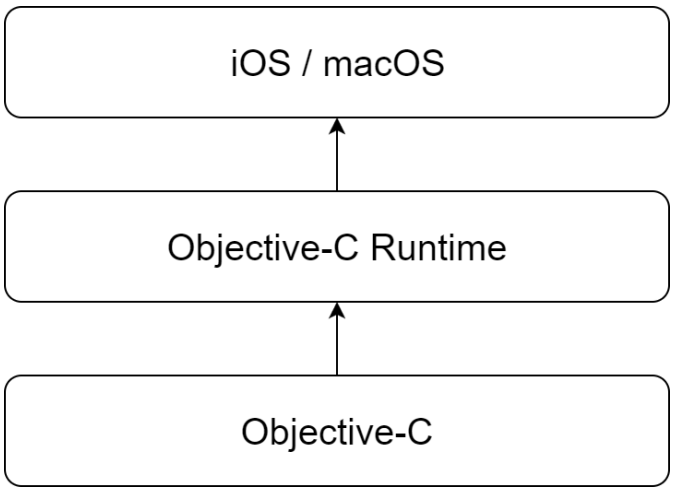


Figure 25: Objective-C Runtime

Introspekce je schopnost objektů (programovacího jazyka) prozradit o sobě informace při běhu programu (runtime). Informace mohou být například: umístění v hierarchií dědičnosti, atributy, jestli implementuje danou metodu, nebo jestli implementuje daný protokol (interface). Jedná se tedy o získání informací o objektu při běhu programu.

Naopak intercession je schopnost objektů (programovacího jazyka) manipulovat s schováním a stavem při běhu programu. Můžeme tedy měnit hodnoty atributů v objektu, přidávat nové atributy, nebo metody. Jedná se tedy o měnění struktury, respektive jeho chování a stavu, objektu při běhu programu.

6.3.2 Ukázka použití

Jedna z nesložitějších části v implementaci bylo využití samotného iOS Runtime API k dosažení požadovaného výsledku. Abychom vůbec mohli Runtime API používat, tak je potřeba nejdříve zmapovat dostupné frameworky, které Runtime API nabízí. Jelikož iOS je uzavřený systém, tak většina části iOS systému je uzavřena a nezdokumentována. Díky Runtime API lze ale najít frameworky, které jsou v systemu dostupné, ale klasickou cestou nepřístupné.
K načtení a zobrazení frameworků z Runtime slouží volně dostupný program pro iOS a macOS nazývaný RuntimeBrowser [26], který byl vytvořen komunitou. RuntimeBrowser zobrazuje dostupné frameworky a třídy, které daný framework obsahuje. U tříd jsou zobrazeny pouze jejich hlavičkové soubory a tak lze vidět, co daná třída nabízí. RuntimeBrowser umožňuje přímo z aplikace vyvolávat dané metody ve třídách z daného frameworku. Takže je možnost testovat a zkoušet použití frameworku za běhu.

Za použití Objective-C Runtime Interospection jsme schopni frameworky použít ve vlastní aplikaci a využít možnosti, které by jinak díky uzavřenosti iOS systému nebyly dostupné.
Jakmile máme nalezený framework a třídu, kterou chceme použít, můžeme začít s vlastní implementací do naší aplikace.
První co musíme udělat je vložit hlavičkový soubor, který jsme získali z Runtime, do projektu. Ten obsahuje danou třídu a všechny její metody a atributy. Jakmile budeme mít instanci dané třídy, tak díky hlavičkovému souboru v projektu nedojde při kompilaci k chybě, že kompilátor nenajde požadovanou třídu.
Druhý krok je vytvoření instance dané třídy a připravení ji k použití.
Poslední krok je samotné využití služeb dané třídy. Díky hlavičkovému souboru třídy, který máme v projektu, můžeme používat instanci třídy běžným způsobem a to například: volat metody a používat atributy.

Následuje ukázka zdrojového kódu v jazyce Swift, který má za úkol vytáhnout z iOS systému všechny nainstalované aplikace, za použití Runtime API.

```
let LSApplicationWorkspaceType = NSClassFromString("LSApplicationWorkspace") as! LSApplicationWorkspace.Type
let applicationWorkspace = LSApplicationWorkspaceType.init()

let installedApp = applicationWorkspace.allInstalledApplications() as! [LSApplicationProxy]
```

—

Kód je velmi jednoduchý a obsahuje jednoduchou část na znázornění využití Runtime API v praxi.

Kód spoléhá na to, že máme v projektu zahrnuté dvě hlavičky z Runtime. A to LSApplicationWorkspace a LSApplicationProxy.

Jakmile to máme, tak můžeme získat odkaz na třídu LSApplicationWorkspace, přes Objective-C Runtime API pomocí metody NSClassFromString(). Návratovou hodnotu přetypujeme natvrdo na třídu LSApplicationWorkspace.

Další krok je vytvořit instanci ze třídy LSApplicationWorkspace. To provedeme velmi jednoduše, a to zavoláním metody init() nad třídou.

Poslední krok je využití nově vytvořené instance. Třída LSApplicationWorkspace, jak jsme zjistili z hlavičkového souboru, obsahuje metodu allInstalledApplications(), která vrací pole instancí třídy LSApplicationProxy, takže při zavolání metody dojde k přetypovaní návratové hodnoty natvrdo na pole instancí LSApplicationProxy. Přetypovávat se musí, jelikož metody vrací neznámy typ návratové hodnoty.

Máme pole instancí třídy LSApplicationProxy a dále můžeme pracovat s objekty a využívat jejich možnosti podle libosti.

6.4 macOS Aplikace

Jedná se o standardní macOS aplikaci využívající systémové prostředky. Nejzajímavějším bodem při implementaci bylo automatické rozpoznání verze souboru které uživatel může nahrávat k projektům. Při rozpoznání verze také dochází k rozpoznání k jakému projektu verze patří a automaticky se k projektu přiřadí. Jelikož rozpoznání probíhá automaticky, tak nedojde k selhání lidského faktoru a nahráni nové verze do jiného projektu. Uživatel má stále možnost změnit rozpoznaný projekt, jelikož v některých případech může ruční zásah být vyžadován. Například při nahráni verze do projektu, který ještě žádné verze neobsahuje, nemůže dojít k automatickému rozpoznání a tak je uživatel vyzván k manuálnímu vybrání projektu.

6.4.1 Rozpoznání verze a projektu

Automatické rozpoznání je založeno na pár předem stanovených podmínkách.

Každá aplikace, která je vytvořena pro iOS systém má jedinečný identifikátor, označovaný jako bundle id. Identifikátor se tvoří jako reverzní název domény a za to další libovolný text. Nejčastěji se přidává název firmy, která aplikaci vyvíjí.

Příklad bundle indetifikátoru. Představme si, že vyvíjíme aplikaci, která se jmenuje MyApp. Bude mít webovou prezentaci na adrese http://www.myapp.com a název firmy, která aplikaci vyvíjí je MyCompany. Výsledný bundle identifikátor bude vypadat následovně:
com.myapp.mycompany. Služba internetu WWW a protokol HTTP se do identifikátoru nedávají. Použití velkých písmen je čistě individuální záležitostí.

Každá aplikace může mít mnoho verzí. Výčet verzí se nejčastěji odvíjí od způsobu vývoje aplikace. Aplikace nejprve prochází debug a aplha verzí, pak beta a nakonec release, respektive store verzi. Takže dostupné verze jsou: Debug, Alpha, Beta, Release a Store.
Podle toho o jakou verzi se jedná je její jméno uvedeno jako sufix v bundle identifikátoru aplikace. Takže předchozí aplikace může mít následující verze a k tomu odpovídající identifikátory:

- com.myapp.mycompany.debug
- com.myapp.mycompany.alpha
- com.myapp.mycompany.beta
- com.myapp.mycompany.release
- com.myapp.mycompany - Verze, která je umístěna do obchodu.

Když uživatel vybere, že chce nahrát novou verzi, tak dojde první k rozparsování souboru a nalezení jeho bundle identifikátoru. Následující krok je nalezení o jakou verzi se jedná. Program v identifikátoru hledá na konci některou verzi z výčtu všech definovaných verzí. Po nalezení správné verze následuje rozpoznání do kterého projektu verze patří. Z originálního identifikátoru se odstraní sufix určující verzi a výsledný podřetězec se hledá ve všech dostupných projektech, jestli neobsahují nějakou verzi s hledaným podřetězcem. Jestli je projekt nalezen, tak je rozpoznán a automaticky vybrán. V případě že není, tak uživatel musí ručně zvolit projekt a při příštím nahráni nové verze už dojde k automatickému rozpoznání podle předchozí verze.

6.5 Prvky architektury

Kromě hlavní architektury použité v aplikacích, se zde nacházejí také návrhové vzory, které řeší konkretní problém.

6.5.1 Composite

Při řešení problému interpretace stromu projektů do datové struktury, byl použit zjednodušený návrhový vzor kompozit. Třída Projekt obsahuje odkaz na svého rodiče, respektive nadřízený projekt. Taky má odkaz na pole svých dětí, respektive subprojektů.
Obecný a konkretní návrhový vzor kompozit.

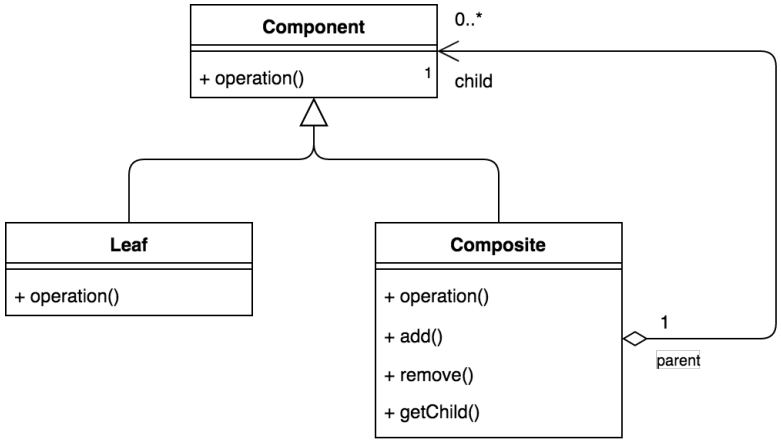


Figure 26: Obecný composite vzor

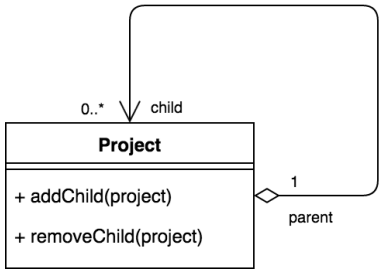


Figure 27: Konrektní composite vzor

6.5.2 Adapter

Použití tříd z Runtime API je někdy velmi nepraktické a nevhodné. Proto jsou všechny třídy, které pochází z Runtime, zaobalené vlastní třídou, která zjednodušuje jejich použití a zanechává čistější kód v celém projektu. Navíc při změně systémových tříd stačí upravit daný Adaptér resp. Wrapper a ostatní kód v projektu bude nedotčen danou změnou.
Obecný a konkretní návrhový vzor adaptér.

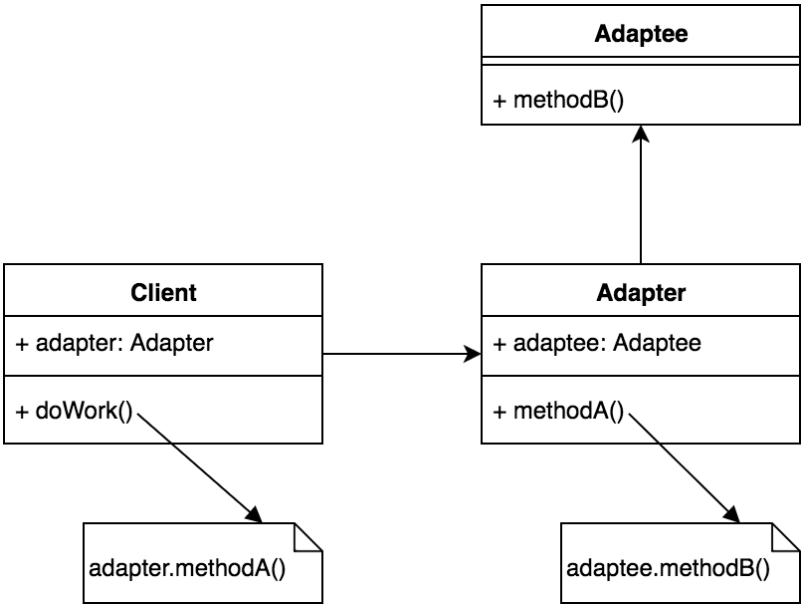


Figure 28: Obecný adapter vzor

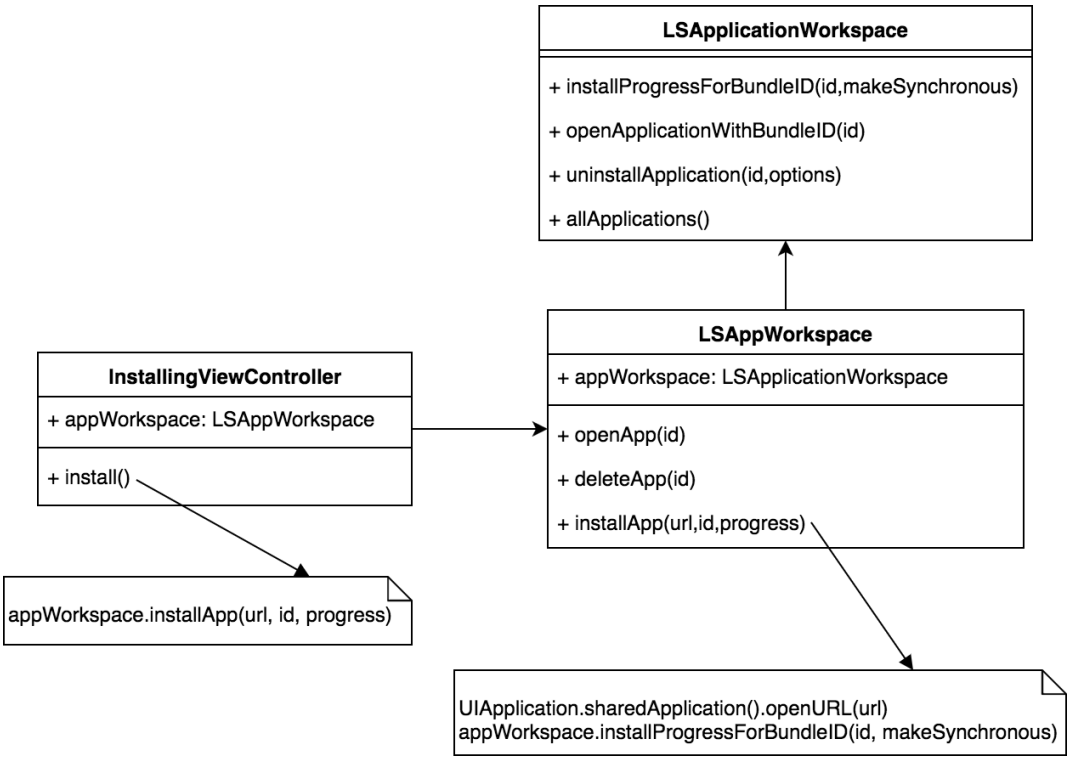


Figure 29: Konrektní adapter vzor

6.5.3 Facade

Při vývoji systému často vyvíjíme nějakou část, která je složitá a obsahuje spoustu implementačních problémů. Použití takové části v ostatních částech systému může být komplikované, proto je potřeba danou část, řekněme subsystém, zaobalit a zpřístupnit přes jednotné a

jednoduché rozhraní. K tomu nám slouží návrhový vzor fasáda.
Jelikož aplikace pro iOS a macOS mají byznys logiku společnou, která je implementována v Dynamic Frameworku, tak veškerá komunikace s frameworkem probíhá přes návrhový vzor fasáda. Usnadňuje to použití frameworku, jelikož všechny jeho třídy zaobaluje jedná třída, která je veřejně vystavena pro použití, ostatní implementace jsou skryté. Při jakékoliv změně v modelu, nebo byznys logiky, nedojde k potřebě zásahu měnit kód v aplikacích využívající framework.
Obecný a konkretní návrhový vzor fasáda.

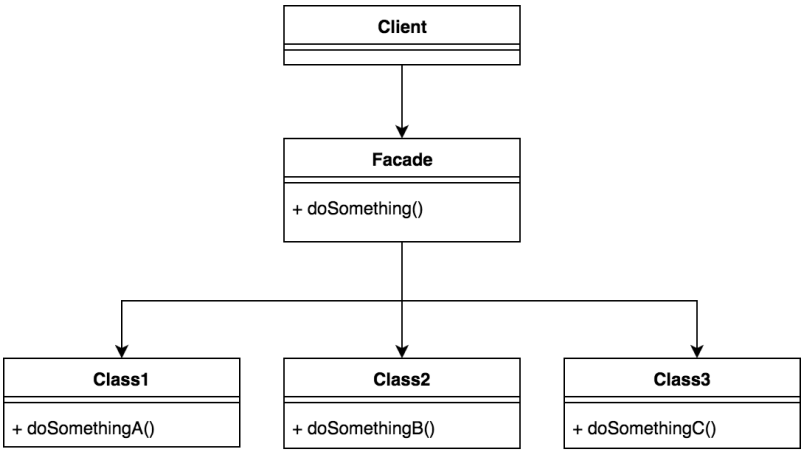


Figure 30: Obecný facade vzor

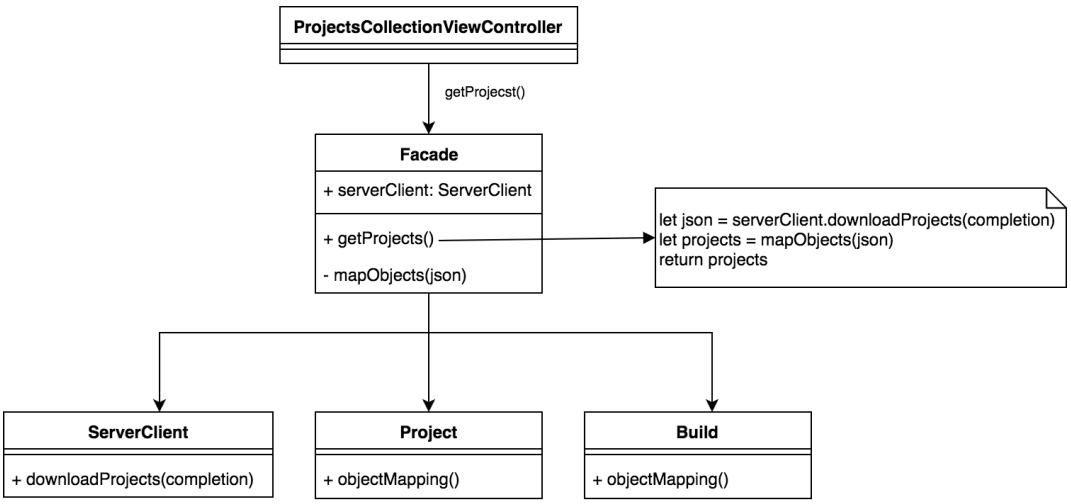


Figure 31: Konkretní facade vzor

6.5.4 Singleton

Někdy při programování v objektově orientovaném jazyce nám stačí pouze jedna instance dané třídy. A ostatní objekty pracují a využívají služby té jedné instance. Tento problém nám řeší návrhový vzor jedináček.
Na iOS platformě se často využívá návrhový vzor jedináček. V rozsáhlé míře to používají

systémové frameworky. Ve vlastní implementaci byl tenhle vzor použit u několika tříd. Například třída, která komunikuje se serverem je implementována jako jedináček. Nebo třída, která se stará o různá nastavení konfiguračních dat.
Obecný návrhový vzor jedináček.

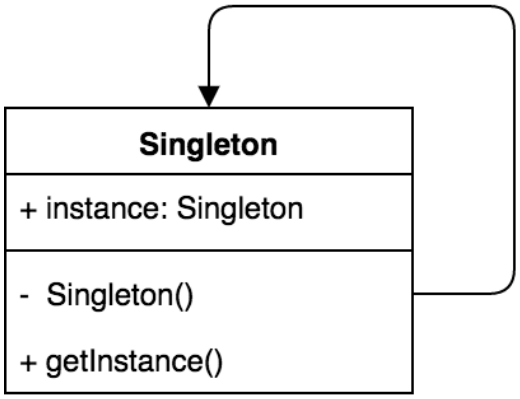


Figure 32: Singleton vzor

6.5.5 Observer

Často potřebujeme řešit problém, že se při změně dat, nebo nějaké akce, provede update uživatelského rozhraní. Řešení problému existuje pod návrhovým vzorem observer. Objekt se zaregistruje k poslouchání nějaké akce a při změně je daná akce vyvolána a objekt je notifikován.
Návrhový vzor, se také používá v systémových frameworcích. Ve vlastní implementaci byl vzor použit hlavně pro update uživatelského rozhraní při změně dat. Například každá obrazovka v iOS aplikaci zobrazuje určitá data a při modifikaci dat je potřeba aktualizovat uživatelské rozhraní. Každá obrazovka se zaregistrovala pro poslouchání dat, které potřebuje a je připravena reagovat na změnu.
Obecný a konkretní návrhový vzor pozorovatel.

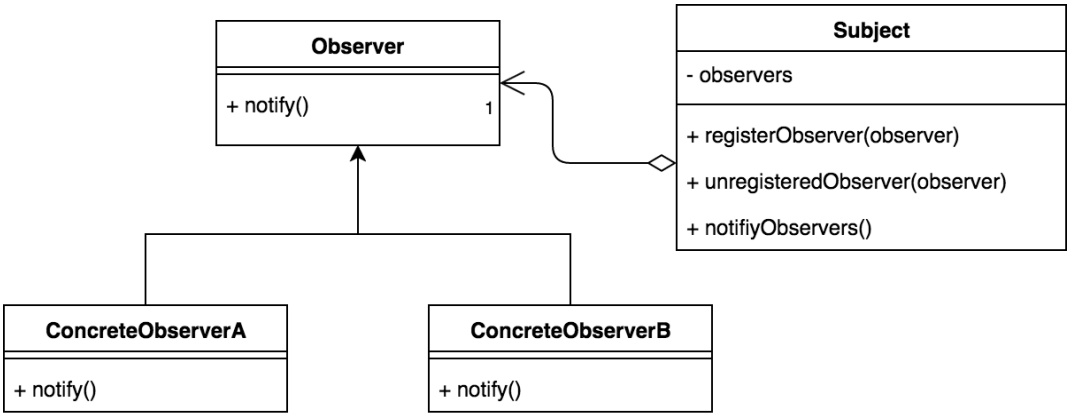


Figure 33: Obecný observer vzor

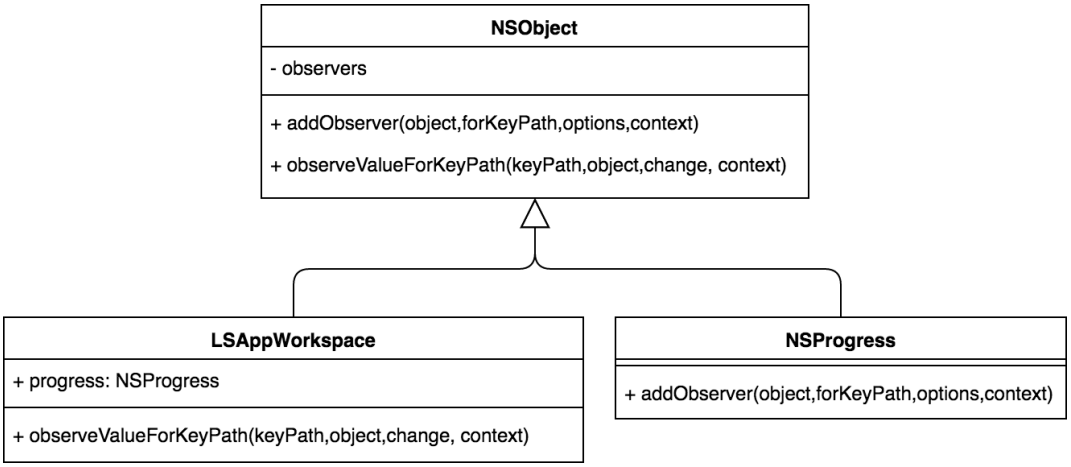


Figure 34: Konkretní observer vzor

7 Testování

Testování software je proces, při kterém dochází k hledání problému, jak už v samotném software, tak případně taky v analýze. Jelikož proces testování spadá pod kategorií ověřování kvality software, tak testovací zaměření může být široké a táhnout se celým vývojem. Při vývoji je možnost testovat několik kategorií, například: unit testy, funkční testy, systémové testy, akceptační testy, atd. Dále budou rozebrány funkční testy, protože byly použity při vývoji iOS aplikace.

7.1 Funkční testy

Jsou to testy, při kterých se ověřuje, že aplikace splňuje všechny požadavky, které byly zaznamenány v analýze. Testují se všechny požadavky a ověřuje se jejich chování vůči analýze. Dále jsou zde nefunkční testy, které ověřují nefunkční požadavky. Pod tím se nachází například výkonové testy, ve kterých se otestuje chování aplikace při velké zátěži. Takové testování jde rozdělit ještě na manuální a automatizované testy.

7.1.1 Manuální testování

Zde dochází k manuálnímu testování aplikace. Prochází se celá aplikace a zkoušejí se všechny její funkce, jestli fungují, tak jak bylo uvedeno v analýze požadavků. Při manuálním testování může dojít k selhání lidského faktoru a tím neprovést testy korektně.

7.1.2 Automatizované testování

Automatizované testy nám urychlují testování. Takové testování muže probíhat buď v předem daných scénářích, nebo v náhodných scénářích. Předem definované scénáře jsou takové, které jsou napsány ručně a dělají nějaký konkretní úkol, například testování přihlašování, které může být po napsání testu spouštěno libovolně mnohokrát za sebou. Náhodné scénáře nejsou předem definované a jejich průběh není přesně stanoven. Je spuštěn nějaký automatizační nástroj, který automaticky ovládá aplikaci.

Pro předem definované testy má Apple ve svém vývojovém prostředí Xcode přímou podporu. Testy jdou psát celé ručně, nebo zachytáváním akcí na obrazovce a tím dochází ke generování kódu daného testu. Takový test potom může být podle potřeby spuštěn. Pro náhodné scénáře existuje nástroj od komunity. Pro iOS a macOS platformu existuje nástroj UI AutoMonkey. Jedná se o skript napsaný v Javascriptu. Skript má mnoho nastavení například: simulace gest, počet kliků, přepínání orientaci obrazovky, atd.

Výsledkem testování je odhalení chyb v software a případně v jeho nekorektním chování. Výstupy z testů jsou dále zpracovány a nalezené problémy opraveny a znova otestovány. Jelikož dochází k opakování stejných testů, a tím ověřování funkčnosti aplikace, tak je v hodné používat automatizované testy, které šetří čas a lidské prostředky.

8 Nasazení

Po implementační fázi je potřeba systém zprovoznit a nasadit do reálného použití. Následně je systém testován, jestli splňuje všechny definované požadavky a chová se korektně za každé situace. Po ověření správnosti systému se přejde do stavu, kdy je systém v praxi používán a udržován.

Vyvíjený systém byl nasazen a v současné době se nachází ve stavu interního testování. V blízké době bude systém odeslán zákazníkům. Tím se celý systém dostane do stavu reálného nasazení, respektive využití v praxi. Během použití systému je jeho stav monitorován a případné chyby, pády aplikací, nebo jiný neočekávaný stav, je analyzován a následně opraven.

Následuje ukázka obrázků z reálné aplikace. Na prvním obrázku je vidět detail projektu, kde jsou dostupné jeho verze a uživatel má možnost provádět dostupné akce. Druhá obrazovka je při výběru instalace aplikace. Zde můžeme vidět průběh a v jaké fázi se instalace nachází.
V prvním obrázku je vidět několik funkčních požadavků, které byly na začátku sepsány. Naopak na druhém obrázku, při instalaci, vidíme využití iOS Runtime API, díky kterému jsme schopni instalovat a sledovat průběh instalace.

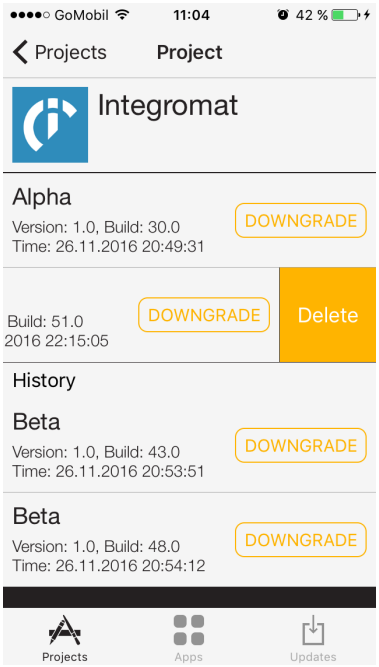


Figure 35: Obrazovka - Project Detail

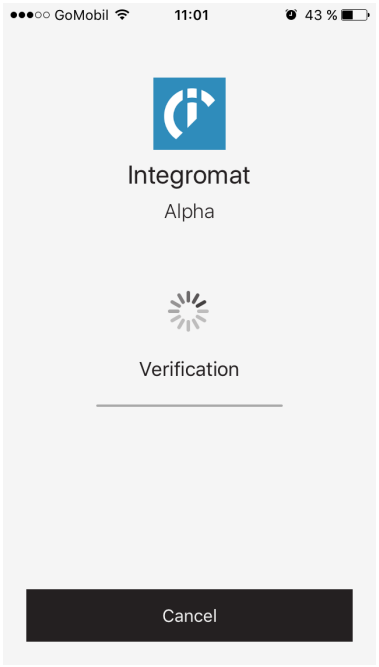


Figure 36: Obrazovka - Install

9 Závěr

Cílem práce bylo vytvoření systému pro distribuci iOS aplikací. Analýzou byly nalezeny nedostatky existujících řešení na trhu. Podle analýzy a požadavků z praxe byly sestaveny případy užití a funkční požadavky. Jako hlavní nedostatky existujících služeb byly nutnost registrace uživatelů do služby, cenová politika, možnost brandování aplikace, nebo neobsahují historií nahraných verzí. Navržený systém tyhle chyby zapracoval do požadavků a následně byly také implementovány a otestovány.

Definované požadavky na systém se podařilo splnit a ověřit jejich správnou funkčnost. Systém byl nasazen a zprovozněn v praxi, kde v současné době probíhá testovací fáze systému. V nejbližší době se počítá s použitím systému u zákazníků.

Dále je už připraven vývoj nových funkci do iOS a macOS aplikací. Například: použití notifikací, update aplikací bez uživatelova vědomí, changelog (popis) ke každé nahrané verzi a mnoho dalšího. Také přidání nových požadavků ze strany zákazníků, které je možné implementovat do systému, díky tomu, že nad systémem je plná kontrola oproti použití existujícího řešení.

Literatura

[1] ROBERT Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. 1 vyd. New Jersey: Prentice Hall, 2008. 464 s. ISBN 978-0132350884.

[2] ERICH Gamma, RICHARD Helm, RALPH Johnson, JOHN Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1 vyd. Boston: Addison-Wesley Professional, 1994. 395 s. ISBN 978-0201633610.

[3] Apple Inc. *TestFlight - Apple Developer* [online]. c2017, [cit. 2017-03-15]. Dostupné z: <https://developer.apple.com/testflight/>

[4] Microsoft. *HockeyApp - The Platform for Your Apps* [online]. c2017, poslední revize 2017-02-18 [cit. 2017-03-15]. Dostupné z: <https://www.hockeyapp.net/>

[5] Alphabet Inc. *The Most Seamless Beta Distribution Experience for You and Your Testers. — Beta by Crashlytics* [online]. c2017, [cit. 2017-03-15]. Dostupné z: <http://try.crashlytics.com/beta/>

[6] Appaloosa. *Features: Private App Distribution, Update and Management — Appaloosa Store* [online]. c2017, [cit. 2017-03-15]. Dostupné z: <https://www.appaloosa-store.com/features?locale=en>

[7] AppBlade. *App deployment, management, and security features — AppBlade* [online]. c2017, [cit. 2017-03-15]. Dostupné z: <https://appblade.com/features>

[8] BirdFlight. *The web-based solution for managing your testers* [online]. c2016, [cit. 2017-03-15]. Dostupné z: <https://www.birdflightapp.com/>

[9] Geek and Tech Corps Pty Ltd. *Installr - Easy iOS and Android beta app distribution* [online]. c2017, [cit. 2017-03-15]. Dostupné z: <https://www.installrapp.com/>

[10] Tutorialspoint. *Django Overview* [online]. c2017, [cit. 2017-03-20]. Dostupné z: <https://www.tutorialspoint.com/django/django_overview.htm>

[11] Apple Inc. *Model-View-Controller* [online]. c2015, [cit. 2017-03-20]. Dostupné z: <https://www.tutorialspoint.com/django/django_overview.htm>

[12] Objc.io. *Introduction to MVVM · objc.io* [online]. c2014, poslední revize 2017-03-07 [cit. 2017-03-20]. Dostupné z: <https://www.objc.io/issues/13-architecture/mvvm/>

[13] Objc.io. *Architecting iOS Apps with VIPER · objc.io* [online]. c2014, poslední revize 2017-03-07 [cit. 2017-03-20]. Dostupné z: <https://www.objc.io/issues/13-architecture/viper/>

[14] Apple Inc. *Xcode - Apple Developer* [online]. c2017, [cit. 2017-01-08]. Dostupné z: <https://developer.apple.com/xcode/>

[15] Carthage. *GitHub - Carthage/Carthage: A simple, decentralized dependency manager for Cocoa* [online]. c2017, [cit. 2017-01-08]. Dostupné z: <https://github.com/Carthage/Carthage>

[16] JetBrains s.r.o. *PyCharm* [online]. c2017, [cit. 2017-01-08]. Dostupné z: <https://www.jetbrains.com/pycharm/>

[17] Bitbucket. *Bitbucket — The Git solution for professional teams* [online]. c2017, [cit. 2017-01-08]. Dostupné z: <https://bitbucket.org/>

[18] Django Software Foundation. *The Web framework for perfectionists with deadlines — Django* [online]. c2017, [cit. 2017-01-08]. Dostupné z: <https://www.djangoproject.com/>

[19] Apple Inc. *Overview of Dynamic Libraries* [online]. c2017, [cit. 2017-01-08]. Dostupné z: <https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/DynamicLibraries/100-Articles/OverviewOfDynamicLibraries.html>

[20] Apple Inc. *address_space1_2x.png* [online]. c2017, [cit. 2017-01-08]. Dostupné z: <https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/DynamicLibraries/art/address_space1_2x.png>

[21] Apple Inc. *address_space2_2x.png* [online]. c2017, [cit. 2017-01-08]. Dostupné z: <https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/DynamicLibraries/art/address_space2_2x.png>

[22] Alamofire Software Foundation. *GitHub - AFNetworking/AFNetworking: A delightful networking framework for iOS, OS X, watchOS, and tvOS* [online]. c2017, [cit. 2017-02-21]. Dostupné z: <http://afnetworking.com>

[23] Lucas Medeiros, Denys Telezhkin. *GitHub - lucasmedeirosleite/EasyMapping: The easiest way to marshall and unmarshall Dictionary representations such as JSON representation* [online]. c2017, [cit. 2017-02-21]. Dostupné z: <https://github.com/lucasmedeirosleite/EasyMapping>

[24] Apple Inc. *Objective-C Runtime - Objective-C — Apple Developer Documentation* [online]. c2017, [cit. 2017-03-28]. Dostupné z: <https://developer.apple.com/reference/objectivec/objective_c_runtime>

[25] Apple Inc. *Introspection* [online]. c2017, [cit. 2017-03-28]. Dostupné z: <https://developer.apple.com/library/content/documentation/General/Conceptual /CocoaEncyclopedia/Introspection/Introspection.html>

[26] Ezra Epstein, Nicolas Seriot. *GitHub - nst/RuntimeBrowser: Objective-C Runtime Browser, for Mac OS X and iOS* [online]. c2016, [cit. 2016-12-05]. Dostupné z: <https://github.com/nst/RuntimeBrowser>

[27] SourceMaking.com. *Composite Design Pattern* [online]. c2016, [cit. 2016-12-20]. Dostupné z: <https://sourcemaking.com/design_patterns/composite>

[28] SourceMaking.com. *Adapter Design Pattern* [online]. c2016, [cit. 2016-12-20]. Dostupné z: <https://sourcemaking.com/design_patterns/adapter>

[29] SourceMaking.com. *Facade Design Pattern* [online]. c2016, [cit. 2016-12-20]. Dostupné z: <https://sourcemaking.com/design_patterns/facade>

[30] SourceMaking.com. *Singleton Design Pattern* [online]. c2016, [cit. 2016-12-20]. Dostupné z: <https://sourcemaking.com/design_patterns/singleton>

[31] SourceMaking.com. *Observer Design Pattern* [online]. c2016, [cit. 2016-12-20]. Dostupné z: <https://sourcemaking.com/design_patterns/observer>

[32] Tomáš Hlava. *Testování softwaru* [online]. c2016, [cit. 2017-4-10]. Dostupné z: <http://testovanisoftwaru.cz/>

[33] Jonathan Penn. *GitHub - jonathanpenn/ui-auto-monkey: UI AutoMonkey is a simple stress testing script for iOS applications that runs in UI Automation and Instruments. Grass fed. Free range.* [online]. c2016, [cit. 2017-4-10]. Dostupné z: <https://github.com/jonathanpenn/ui-auto-monkey>

Seznam příloh

- **Příloha A** - Obsah přiloženého CD, 1 strana.
- **Příloha B** - Návrh uživatelského rozhraní, Příloha na CD/DVD.

10 Přílohy

10.1 Příloha A

10.1.1 Obsah přiloženého CD

```
/
├── source ..... adresář se zdrojovými soubory
│   ├── code ..... adresář obsahující implementaci
│   └── thesislatex.tex ..... zdrojový soubor práce v TEX
└── texts ..... texty práce
    ├── prilohaB.pdf ..... příloha B ve formátu PDF
    └── thesis.pdf ..... bakalářská práce ve formátu PDF
```